# OS-9/6809 ユーティリティ

金井　隆
箕原辰夫

SHUWA
SYSTEM
TRADING
CO.,LTD.

**注　意**

(1)本書は著者らが調査した結果を出版したものです。

(2)本書は内容について万全を期して作成いたしましたが，御不審な点や誤り，記載もれなどお気付きのことがありましたら，出版元まで書面にて御連絡ください。

(3)本書の内容に関して運用した結果の影響については(2)項にかかわらず責任を負いかねますので御了承ください。

(4)本書の全部または一部について出版元から文書による許諾を得ずに，いかなる方法においても複写，複製することは禁じられています。

# OS-9/6809
# UTILITIES

SHUWA SYSTEM TRADING CO., LTD. 秀和システム

# まえがき

近年マイクロコンピュータの分野でもハードウェアおよびソフトウェアの大規模化に伴い、高度なアプリケーションをサポートし得るだけの性能を持ったOSの必要性が認識されるようになってきました。そのような中にあってOS－9はマルチユーザ、マルチタスクをサポートするOSとして、特に注目を集めています。

OS－9は16ビット、32ビットOS　UNIXを手本にしているため階層ディレクトリやパイプラインなどが自然な形で導入されており、さらにプログラムモジュールの概念など今までに無い使い勝手の良さを誇っています。

しかし、ハードウェアの限界は仕方のないこととして、アプリケーションプログラムの量においては、今だUNIXには及びません。それは、UNIXがリダイレクトやパイプラインといった方法によって過去のプログラムの蓄積を利用できるような環境を用意しており、その上長い年月にわたって多くの人手により洗練されてきたためです。にもかかわらずOS－9は、比較的小規模のシステムではUNIXを凌ぐ性能を示すことがあります。

本書は基本的なコマンドおよびフィルタ群を揃えることから環境の整備を図ることを目的としています。

これらユーティリティの多くは、筆者らが実際日常的に使用しているものです。

第1章ではコマンドおよびフィルタを作成する際に考慮すべき点を簡潔にまとめました。

第2章ではコマンド記述に最も向いていると言われるC言語によるプログラムをいくつか挙げました。マイクロウェアの提供するCはフルセットのCですから、他のOSへの移植も容易なはずです。またマイクロウェアのPascalを使ったものも参考として掲載しました。

第3章ではBasic09を扱います。Basic09はOS－9に付属の言語であり、他に類を見ない性能を持っておりますが、コマンド記述という視点から見ると不都合な点がいくつかあります。この章ではそれらを補う方法を提案します。

第4章はOSとユーザの橋渡しをするShellについて述べました。Shell自身はコマンドであって同時にフィルタであるということを思い出して下さい。またShellの機能拡張についても触れます。

なお、本書の作成にあたって印南比呂志、岩瀬浩治、平田幸広の各氏にご協力いただきました。

本書によってOS−9がより使い易いものとなり、また多くのアプリケーションが産みだされることを切に希望します。

金井　隆

１９８４年１１月

箕原辰夫

[参考文献]

『プログラミング言語Ｃ』
B.W.Kernighan　D.M.Ritchie著　共立出版

『ソフトウエア作法』
B.W.Kernighan　P.J.Plauger著　共立出版

『Pascalの標準化』−ISO規格全訳とその解説−
共立出版

『OS−9　Ｃコンパイラユーザーズマニュアル』
マイクロウェア

『OS−9　Pascal V2.0ユーザーズマニュアル』
マイクロウェア

『ＰＡＳＣＡＬプログラミング　補遺１　構文図式』
W.Findlay　D.A.Watt 著　日本コンピュータ協会

『アルゴリズム＋データ構造＝プログラム』
N.Wirth著　日本コンピュータ協会

『ＦＭ−11　Ｆ−BASIC文法書』
富士通

『FM−7　Ｆ−BASIC解析マニュアル　フェーズII』第５章
箕原辰夫、南泰浩、菊地寿共著　秀和システムトレーディング（株）

『OS−9　Level 1　解析マニュアルＩ』
金井隆著　秀和システムトレーディング（株）

# Contents

## コマンドの設計

1—1　言語　3
　1—1—1　簡単なコマンド　3
　1—1—2　Shellプロシジャ　4
　1—1—3　Basic09　6
　1—1—4　C　8
　1—1—5　アセンブラ　10
　1—1—6　言語の選択　12
1—2　コマンドパラメータ　13
　1—2—1　パラメータの区切り　13
　1—2—2　ファイル名とオプション　14
　1—2—3　オプション無しの場合　15
　1—2—4　オプションフラグ　17
1—3　コマンドの組み合わせ　23
　1—3—1　パイプラインの応用　23
　1—3—2　グルーピングの応用　24

## Cによるプログラム

2—1　C概設　29
　2—1—1　argcとargv　29
　2—1—2　入出力　30
　2—1—3　文字列　31
2—2　プログラム例　32
　2—2—1　join　32

2—2—2 sep 33
2—2—3 place 35
2—2—4 head 38
2—2—5 tail 40
2—2—6 uniq 44
2—2—7 apart 45
2—2—8 dmp 50
2—2—9 fold 55
2—2—10 fed 57
2—2—11 cp 69
2—2—12 calcu 78
2—2—13 cal 105
2—2—14 lprint 110
2—2—15 grep 116

## Basic 09によるプログラム

3—1 パラメータの渡し方 127
3—1—1 I-Codeモジュールへのパラメータ 127
3—1—2 括弧をはずす 128
3—1—3 パラメータの解釈 131
3—1—4 プロシジャ名の衝突 134
3—2 プログラム例 136
3—2—1 err 136
3—2—2 mh 137
3—2—3 para 144
3—2—4 time 149

3—2—5　at　153
3—2—6　wild　156
3—2—7　find　163
3—2—8　smem　168
3—2—9　flink　170
3—2—10　bprep　179
3—3　グラフィック　181
3—3—1　gcursor　181
3—3—2　circle　185
3—3—3　sound　193
3—3—4　geditor　194
3—3—5　cards　211

## Shellの拡張

4—1　コマンドおよびフィルタとしてのShell　223
4—1—1　フィルタとしてのShell　223
4—1—2　Shellオプション　224
4—2　コマンドライン解釈の手順　225
4—2—1　セパレータ　225
4—2—2　リダイレクト　226
4—2—3　グルーピング　229
4—2—4　コマンドの形式　231
4—2—5　コマンドの取出し　234
4—3　セパレータの処理　237
4—3—1　シーケンシャルセパレータ　237
4—3—2　コンカレントセパレータ　237

4—3—3　パイプラインセパレータ　237
4—4　コマンドの起動　241
4—4—1　モジュールのサーチの順序　241
4—4—2　内蔵コマンド　244
4—5　プロトタイプShell　247
4—5—1　パターンマッチ　247
4—5—2　wsh　249
4—5—3　拡張Shell　253

## 付録

mcat　277
chmod　277
ls　281
ln　286
rm　289
find　291

# コマンドの設計

# 1－1　言語

## 1－1－1　簡単なコマンド

手始めに簡単なコマンドを考えてみましょう。画面を消去するclearです。

OS9：clear

これだけを見て、clearがいったいどんな言語で記述されているかわかるでしょうか。dirかdumpでもしない限りそれを知る手だてはありません。

つまり、ユーザがコマンドを使う時は、それがどんな言語で書いてあるかなどということを知っている必要はないのです。ユーザが知っていなければならないことは、「コマンドを起動すると何が起きるか」という事実と、「どうやって起動すればいいのか」という2点です。

コマンドを作ろうとする際に最も重視すべきなのはこの点です。

では実際にclearを設定してみましょう。

画面を消すためには、$0Cを「標準出力」へ書き出すという方法をとります。/termをオープンするようなことはしません。標準出力を使うのは、例えば出力をプリントにリダイレクトできるという理由からだけではありません。もし端末を使っているとしたら、もはや/termをオープンして$0Cを書き出したとしても端末の画面を消去することにはなりません。

ではclearを記述する言語を決めましょう。先に述べたようにコマンドの働きが同じであれば、どんな言語を使おうと同じことです。同じなら、なるべく簡単な言語を使いたいものです。一般的に易しい順に並べてみますと、

1．Shellプロシジャ
2．Basic09
3．C（またはpascalなど）
4．アセンブラ

となります。しかし、実際にはコマンドの働きが同じでも、実行環境、スピード、メモリ効率などが違います。

### 1－1－2　Shellプロシジャ

clearをShellプロシジャで記述します。

```
display c
```

このように簡単に書けてしまう理由は、すでにdisplayというコマンドがあるからに他なりません。つまり、過去に作成済みのコマンドを使って新しいコマンドを作ることができるわけです。これは「ソフトウェアの蓄積」という観点から見た時に強力な武器となります。 しかしながら現在のShellの機能が(UNIXのそれと比較して)低いため、Shellプロシジャの使用にあたってはいくつかの制約が付きます。

1．プロシジャはカレントディレクトリになければならない。
2．プロシジャには引数が渡せない。
3．中で使っているコマンドが存在しなければならない。
4．他のコマンドを起動するためオーバーヘッドが大きい。
5．メモリモジュールに置くことができない。

まずShellプロシジャはカレントデータディレクトリに存在しなければなりません。これはShellがコマンドを次のような順で探すためです。(完全なパスリストで与えた場合は除きます。)

1．モジュールディレクトリ
2．カレント実行ディレクトリ
3．カレントデータディレクトリ

実行ディレクトリにあるファイルは実行可能(アトリビュートの'e'ビットがオン) でなければならず、逆にデータディレクトリのファイルは実行可能であってはなりません。

したがって。Shellプロシジャは実行ディレクトリに入れておいても意味がありません。一度データディレクトリが他に移ってしまうと、完全なパスリストを与えないと起動できないことになり、大変に不便です。

これを避けるためにはShellの若干の改良が必要になります。さもなければ真にコマンドとして使うことはできないでしょう。したがって、作業用の（テンポラリな）コマンドとして用いるか、または他の言語でコマンドを書く前のプロトタイプとして用いるのがよいでしょう。

Shellプロシジャへは引数を渡すことができません。コマンドとして使用するためにはこの点が妨げとなります。これはShellを改良する以外に手が無く、そう簡単な作業ではありません。

Shellプロシジャを起動する時には、プロシジャの中で起動するコマンド（例えばclearではdisplay）が無ければ動きません。このことは意外に忘れがちです。もしwsetを削除すると、もはやwindow2を実行することはできません。

Shellプロシジャはその中で他のコマンドを起動するわけですから、そのコマンドがどこにあるかによってプロシジャの使い勝手は良くも悪くもなります。

clearを例にとると、もしdisplayがメモリモジュールにあればdisplay起動のオーバーヘッドはありませんが、一般には実行ディレクトリに存在し、そこからロードしなければなりません。clearのようにdisplay 1つだけを1回しか起動しないのならまだ我慢できますが、次の場合には大変なことになります。

```
OS9：wild  ＊  ！ place  copy  ＊  /r0/＊  ！ shell  －p
```

wildがファイル名を一つずつ送る度にplaceはcopyをロードします。ファイルがたくさんある場合にはcopyのロードのために多くの時間を浪費するでしょう。しかし、copyが前もってロードしてあればその時間のロスはありません。このことを考慮して、dsaveコマンドでは最初にcopyをロードするようにしていますが、逆にcopyがすでにロードしてある場合、すでにロードしてあるにもかかわらずロードし直すという無駄が生じます。

Shell プロシジャはモジュールではないため、モジュールディレクトリにロードすることができません。したがってプロシジャを起動する時は必ずディスクからロードするという手間がかかります。

以上から、同じコマンドといっても Shell プロシジャの場合にはいろいろと不都合があります。それにもかかわらず、Shell プロシジャが有効であることは言うまでもありません。

## I—I—3　Basic09

clear を Basic09で記述します。

```
PROCEDURE clear
dim c: byte

    c = $0c
    put #1, c
end
```

コーディングは比較的簡単であり、また（比較的）わかりやすく、特にデバッグが容易であることを強調しておきます。

しかしよいことばかりではなく、コマンド記述という面から考えた場合はいろいろと欠点が目につきます。(Pack したものを考えています。)

1．Runb が無ければ動かない。
2．コマンドパラメータを渡すのが大変。
3．１つのプログラムが複数のモジュールから構成される。
4．言語仕様上の問題。

Basic09はインタープリタであるため、I - Code（一種の中間言語）を実行するためのランタイムルーチンである Runb が必要です。と言うより、走るのは Runb の方であって、I - Code は Runb に対するデータにすぎないと考えた方が正しいでしょう。

Basic09で作ったコマンドがいくつかあるのならRunbをメモリに常駐させておくべきですが、レベル1のようにメモリが少ないシステムでは問題です。メモリ全体のうちRunbの占める割合が小さければ、システム全体としてのメモリ効率が向上します。

Basic09のプログラムにパラメータを渡す場合は、プログラム中と同様に括弧でくくらなければなりません。文字列を渡す時はダブルクォートでかこむ必要があります。そのため、コマンドとしてはアセンブラやCで記述したものと別の扱いをしなければなりません。

すなわち「コマンドパラメータの個数およびその順序、型をユーザが覚えていなければならず」、大変な苦痛だと言えます。また、「そのコマンドがBasic09のプログラムということを覚えていなければならない」ということは、それ以上の苦痛です。

しかしこの問題は比較的簡単に回避することが可能です。具体的な方法は3章で述べます。

Basic09のプログラムは一般には複数のI-Codeモジュールから構成されます。すなわちコマンドとして起動すると複数のモジュールがメモリモジュールに登録されるわけです。直接見たくないPROCEDUREまでもがモジュールとして陽に存在してしまうのです。

レベル1では個々のモジュールに対して256バイト単位のメモリが与えられるのでメモリが無駄になってしまいます。(レベル2ではパッケージ化しておけば無駄は生じませんが。)

Basic09のモジュール化はもう1つの問題を生じます。実行中にコントロールEなどを用いて中断しようとした時、あるいは何らかのエラーで中断した時、いくつかのモジュールがメモリ上に残ってしまうことがあります。これは、中断した時点でのリンクカウントがゼロでないモジュールは削除されないからです。ただし、メモリに常駐させておくつもりなら問題はありません。

さて、Basic09はもともとシステムやコマンドの記述のために作られた言語ではないので、それをコマンド記述のために使うにはいろいろと無理があります。モジュール化を徹底させるためにグローバル変数が使えない、記号定数が使えない、関数を増設できない等々です。

## 1—1—4　C

Cは高級アセンブラと異名をとる位に記述性にすぐれており、また数ある言語の中で最も移植性が良いことは、多くのシステムで実際にＵＮＩＸが動いていることからも実証されています。

CのプログラムはあらゆるＯＳ（ＯＳ９／６８０００、ＵＮＩＸ、ＭＳ－ＤＯＳ等々）へ移植することが容易であり、その逆もまた容易です。(ほとんどのプログラムは何の変更も必要としません。)

clear をCで記述します。

```
#include <stdio.h>

main()
{
    putchar( 0x0c );
}
```

Cは分割コンパイルの考え方が徹底しており、一度作った関数はそのまま利用できます。また仕様が同じならばその関数を取り替えても他に影響はありません。例として、ファイルを削除する関数を書いてみます。(標準ライブラリの unlink と同じものです。)

```
delete( p )
    char *p;
{
    char q[81];

    strcpy( q, "del " );
    strcat( q, p );
    system( q );
}
```

system 関数を使うことによって比較的簡単に delete を作ることができました。同様に、system を使えば（コマンドがあるものは）すべて実現できます。この delete.c をリンカにかけてリロケータブルオブジェクトに落とし、ライブラリ clib.l に追加（merge するだけ）しておけば以後は標準関数とまったく同じに使用できます。プロトタイプとしてはこれで十分ですが、del 呼び出しのオーバーヘッドを避けるために_os9関数を使ったものを作ってみます。

```
#include <os9.h>

int delete( p )
    char *p;
{
    struct registers reg;

    reg.rg_x = p;
    return( _os9( I_DELETE, &reg ) );
}
```

この場合一つだけ注意しなければならないのは、systemは標準関数であっても_os9はそうではないということです。

標準関数と同じようにdeleteを使いたいのならば、最初からアセンブラで記述しておくという方法もあります。

```
#include <os9.h>

#asm
delete:
 ldx 4,s
 os9 I_DELETE
 bcs del1
 clra
 clrb
 bra del2
del1
 clra
 std errno,y
  ldd #-1
del2
 rts
#endasm
```

```
 psect delete_c,0,0,0,0,0
 nam delete_c
delete:
 ldx 4,s
 os9 I_DELETE
 bcs del1
 clra
 clrb
 bra del2
del1
 clra
 std errno,y
  ldd #-1
del2
 rts
 endsect
```

左はCのソースとしてコンパイルする場合、右はアセンブラc. asmのソースとしてアセンブルする場合です。

以上のいずれを用いても同じ機能の関数ができます。

さて、Cにも若干の問題点があります。

1．コンパイルに時間がかかる。

2．プログラムサイズが大きくなる。

コンパイルにだいぶ時間がかかります。これは、プリプロセッサからリンカまで含めて7パス程度あり、パスごとに中間ファイルを作っているためです。Cの

機能性、柔軟性の代償と考えればしかたのないことでしょう。ちなみにオンメモリ（レベル２でＣを全部ロードしてかつＲＡＭディスクを使用）で数百行程度のソースなら約30秒から50秒かかります。フロッピベースなら約4—5分程度かかるでしょう。

プログラムサイズが（アセンブラで書く場合と比べて）かなり大きくなりますのでレベル１のようにメモリの少ないシステムでは問題があるかもしれません。Ｃのスタートアップルーチン cstart が無条件で置かれることと、標準関数がかなり大きいことが原因でしょう。

例えばアセンブラなら数百バイトで済むものがＣだと数キロバイトになってしまいます。レベル１のように、メモリの少ないシステムではプログラムサイズは深刻な問題かもしれません。

しかしプログラムサイズの問題以上に問題なのはプログラム作成のための手間と、（さらに重要なのは）「保守」および「変更」のための手間です。これは何も難しいことではありません。昨日作ったプログラムが今日理解できないといった事態を避けるにはどうしたらよいか、ということです。もし「このコマンドのオプションを１個増やしたい」と思った時、Ｃなら簡単にできますが、アセンブラでは多くの労力と暴走の危険を覚悟しなければなりません。

## 1—1—5　アセンブラ

アセンブラで作ったプログラムは、速い、小さい、メモリを食わないプログラムを書き得るという点では高級言語を寄せつけません。反面、開発が難しい、解読が困難、保守や変更が大変だという欠点を持っています。特にアセンブラプログラムの開発には暴走の危険がつきまとうため、マルチユーザ環境でデバッグすると他のユーザに迷惑がかかります。これはハード的な保護がないシステムでは仕方のないことです。

また、FMシリーズのOS—９／６８０９に付属の言語はアセンブラと Basic09の２つですが、Basic09ではシステムに近い部分の記述が不可能なため、アセンブラを使用せざるを得ない場合があります（例えばグラフィック関係がそうです。）。ただし、Ｃを使って Basic09のサブルーチンモジュールを作ることはできます。

さて、アセンブラでclearを記述してみます。

```
                *****************************************
                *
                *    clear  --  clear screen
                *
                
                                        ifp1
                                        endc
                
0000 87CD002D                           mod    Size,Name,Prgrm+Objct,ReEnt+1,Entry,Mem
000D 636C6561           Name            fcs    /clear/
0012 01                                 fcb    1           edition number
00C8                    Mem             equ    200

0001                    STDOUT          equ    1

0013                    Entry           equ    *
0013 860C                               lda    #$0c
0015 3402                               pshs   a
0017 1F41                               tfr    s,x
0019 108E0001                           ldy    #1
001D 8601                               lda    #STDOUT
001F 103F8A                             os9    I$Write
0022 3261                               leas   1,s
0024 2501                               bcs    exit
0026 5F                                 clrb
0027 103F06             exit            os9    F$Exit

002A 0B0C6E                             emod
002D                    Size            equ    *
                                        end
```

少し考えれば、clear程度のコマンドならば実行速度を気にすること自体が無意味だということが分かるでしょう。またオブジェクトを１バイト減らす工夫をしてもほとんど意味がありません。実際のコマンドの類はこれと同様に、スピードやサイズに神経質になる必要がないのがほとんどです。

より重要なのは、デバッグ終了までに要する時間と労力です。アセンブラをあえて使用するのは、それにみあうだけのものが完成するという見込みが無い限りは避けた方がよいでしょう。そしてもう一つ忘れてはならないのが、仕様や機能の変更あるいは拡張が容易であるかどうかです。

## 1－1－6　言語の選択

いままでの議論で、どの言語がどのような長所と短所を持ち、どのような点に気をつければよいかが分かってきたでしょう。それらを踏まえた上で、どの言語を選んだらよいかを考えることにします。コマンドおよびフィルタを作ることを前提にしていることを忘れないでください。

まず頭に置かなければならないことは、ＯＳ－９は大変に柔軟性のあるＯＳであるということです。つまり、メモリを増やせば増やす程、また高性能のディスクを接続すればよりいっそう使い易くなるのです。この点は他のマイコンやミニコンのＯＳには見られない持色です。

したがって、システムを何に使うかによって、または自分のシステムがどの程度のものであるか、あるいは将来どの程度にするつもりかによって、コマンドの設計方針が変わってくる可能性があるのです。

レベル１でフロッピベースのシステムだと、メモリが最も貴重な資源ということになり．プログラムサイズを小さくすることが最重要課題となります。Ｃではプログラムサイズが大きくなるし、またRunbを常駐させることが難しいので結局アセンブラに頼らざるをえません。

レベル２でメモリに余裕があるシステムだと、Runbを常駐させることができますのでコマンドをBasic09で記述でき、トータルでのメモリ使用量を小さくできる上、実行速度とスループットも十分なものができます。

主要なコマンドをメモリ上に置いておくことができれば、Shellプロシジャやパイプラインの実行の際にいちいちロードしなくても済むので、応答速度が速くなります。

また、もしＯＳ－９／６８０００や他のＯＳを使ってみるつもりならプログラムの移植性を最重視しなければならず、アセンブラはもはや使えません。そのためＣを使うことになるでしょう。

以上のような理由で、何を基準にしてコマンド設計をすべきかが問題ですが、本書では開発環境を整備することを最重要課題としているため、記述性、移植性および保守性の悪いアセンブラを使用するのを避け、Ｃに主眼を置いています。

# 1－2　コマンドパラメータ

## 1－2－1　パラメータの区切り

コマンド（フィルタも含めて）には通常何らかのパラメータを与えます。パラメータとして最も多く使われるのはファイル名です。

```
OS9：list　file1　file2　file3
```

とした場合、'file1'と'file2'と'file3'という３つのパラメータが与えられたものと解釈できます。３つの、と分かるのは空白を「パラメータの区切り」と見なしているからに他なりません。

```
OS9：west　0, 0, 80, 25
```

の場合はコンマを区切りと見なしています。

実は、これらのコマンドに渡されるのは「区切り」も一緒にした「唯一個の文字列」ににすぎないのです。「区切り」を「区切り」として解釈しているのはコマンド自身なのです。この例ではwsetには'0, 0, 80, 25'という文字列が渡されます。

コマンドはまずパラメータを解釈することから始めなければなりません。wsetの中身は、パラメータの解釈がほとんどを占めています。

ＯＳ－９のたいていのコマンドは、空白とコンマの両方ともにセパレータであると見なすものが多いのですが、空白しか認識しないものも多くあります。例えば、wsetは両方を区切りとし、listは空白のみを区切りとします。

Ｃでは自動的に空白とコンマをセパレータと見なします。関数mainにパラメータが渡されたときには、すでに区切られた文字列が渡されますので、コーディングの際には気にする必要はありません。Basic09では後で述べるように、自分で区切る作業が必要となります。

空白を区切りのためのキャラクタとして使ってしまうと、文字列の中に空白を含めたい場合に困ります。その時はダブルクォートまたはシングルクォートで文字列をくくることにしましょう。ただしダブルクォートで始まった文字列は、再びダブルクォートが現われる場所かＥＯＬ（end of line）までとし、中にシング

ルクォートを含んでいても構いません。その場合シングルクォートは区切りの意味を失います。シングルクォートで始まる場合にはダブルクォートが意味を失います。

これはＯＳ－９のＣが行っている解釈の方法ですが、本書では言語を問わずにこの解釈に従うことにします。

問題は、このような方法をもってしても、Shellの予約キャラクタをパラメータの中に含めることができないということです。（；＆！＃）

Shellコマンドラインの解釈の際には、単に１バイトずつ予約キャラクタを捜してゆき、そこまでを１つのコマンドとしてしまうからです。これはShellを改造しない限りはなんともなりません。

## 1－2－2　ファイル名とオプション

コマンドに対するパラメータは、ファイル名であることが最も多いですが、それ以外に、例えばdirコマンドの”ｅ”オプションのように、コマンドに対する細かい指示である場合があります。

以下、ファイルおよびディレクトリ名のことを「ファイル名」、それ以外のコマンドに対する指示のことを「オプション」と呼びます。

オプションは通常省略が可能であり、省略したときには何等かの省略値が与えられるようにコマンドを設計します。

さてコマンドパラメータとしてファイル名とオプションの両方が与えられた場合、ファイル名であるかオプションであるかを区別する方法がなければなりませんが、それには２つ考えられます。

まず、パラメータを置く位置で区別する方法、例えばdirでは、

```
OS9：dir   tempdir   e
```

のように、第一パラメータとしてディレクトリ名を置き、第二パラメータとして”ｅ”オプションを置くと、固定的に決めてしまうことです。この場合、

```
OS9：dir   e   tempdir
```

のようなものは許されません。

この方法だと、パラメータの順番を正確に覚えている必要が生じ、また、オプションの省略がある場合に解釈が難しくなります。

次にオプションのために特別のキャラクタまたは文字列を使う方法。dir の例では”e”をオプション用の特別の文字列として、”e”をどこに置いてもよいことにします。しかし、”e”を特別のものとしますと、もし”e”というディレクトリがあったとしてもそれに dir コマンドを適用することができません。

もし特別のキャラクタとして、ファイル名の中に含んではいけない文字を選べば問題は起きません。そのためにＵＮＩＸからの伝統で、オプションを表すキャラクタとして’−’を使用します。オプションの種類が複数ある場合は’−’の次に種類を表すアルファベットを置くなどします。こうしたオプションを特に「オプションフラグ」と呼びます。

## 1−2−3　オプション無しの場合

オプションの解釈の話の前に、オプションの無いコマンドを考えましょう。ファイルの行数を数える lc コマンドを例にとります。

まず純粋にフィルタとしてのみ使用できるバージョンです。

```
/*    lc  --  only filter version     */

#include <stdio.h>

main()
{
    char line[BUFSIZ];
    int  n;

    for ( n = 0; gets( line ); n++ )
        ;
    printf( "%05d¥n", n );
}
```

パラメータを使わないので main では argc と argv を宣言していません。入力が標準入力に固定的であるために１行読み込みには gets を使っています。

次に純粋にコマンドとしてのみ使用でき、フィルタとしての機能を持たないバージョンです。

```
/*    lc  --  only command version    */

#include <stdio.h>

main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();

    while ( --argc ) {
        if ( ( fp = fopen( *++argv, "r" ) ) == NULL ) {
            fprintf( stderr, "can't open %s.¥n", *argv );
            return;
        }
        lc( fp );
        fclose( fp );
    }
}



lc( fp )
    FILE *fp;
{
    char line[BUFSIZ];
    int  n;

    for ( n = 0; fgets( line, BUFSIZ, fp ); n++ )
        ;
    printf( "%05d¥n", n );
}
```

1つのファイルについて行数を数える関数をmainから切り離してlcと名付けました。関数lcはfpで表されるファイルから入力しますが、fpがstdinの時は最初の例と同じことになります。またgetsはfgetsにしなければなりません。

ではこの2つの考え方をまとめて、コマンドとしてもフィルタとしても使えるように拡張してみましょう。つまり、パラメータとしてファイル名が1つも置かれなかった場合はフィルタとして標準入力から読み込み、ファイル名が1つ以上置かれた場合はそのファイルから読み込みます。

ファイル名を指定された場合はオープン、クローズの処理が必要であるのに対し、標準入力の場合はその必要が無い（というよりオープン、クローズをしてはいけない）という違いがあります。したがって全体として2つの流れに分けられることになるのです。この考えはあらゆるコマンド（あるいはフィルタ）の基本です。

```
/*    lc  --  line count  */

#include <stdio.h>

main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();

    if ( argc == 1 )
        lc( stdin );
    else
        while ( --argc ) {
            if ( ( fp = fopen( *++argv, "r" ) ) == NULL ) {
                fprintf( stderr, "can't open %s.¥n", *argv );
                return;
            }
            lc( fp );
            fclose( fp );
        }
}



lc( fp )
    FILE *fp;
{
    char line[BUFSIZ];
    int  n;

    for ( n = 0; fgets( line, BUFSIZ, fp ); n++ )
        ;
    printf( "%05d¥n", n );
}
```

このmainは、オプション無しのたいていのコマンドに使えます。lcを呼んでいるところを、然るべき関数を呼ぶように変更するだけです。

## 1—2—4　オプションフラグ

オプションフラグを使うとパラメータの解釈が楽になります。パラメータの一つひとつについて、その第1バイト目を見ればオプションであるかないかの判定ができるからです。例として、listと同じことをするコマンドであるcatを作ります。まったく同じでは面白くないのでオプションを2つ付けましょう。出力する各行に行番号を付ける'−n'と、1つのファイルを出力した後に改ページする'−f'です。'−f'はプリンタに出力の際に用紙の頭を合わせるために付けたものです。

オプションをどこに置くかで2つの考え方があります。

一つは、オプションはファイル名よりも前に無ければならないという制限を設けることにより、オプションの解釈を簡単にしようというものです。

OS9：cat　−n　−f　file1　file2

関数catはファイル1つの処理をします。mainを見て下さい。whileを抜けた時、argcは引数の数からオプションを引いた数を示しています。それがゼロであれば引数としてファイル名は1つも無かったということになりますので、標準入力を割り当てます。そうでなければ残っているファイル名の個数だけwhileループを実行します。

外部変数であるnumberとformfeedにオプションのＹＥＳまたはＮＯの値を設定します。この2つをグローバル変数としたのは、mainだけでなくcatも参照するからです。main内のローカル変数として宣言しておいて、catには引数として与える手段もありますが、グローバルにした方が簡明であろうことと、後あとのことを考えて外部変数にしました。このへんはだいたい趣味の問題ですので、自分で納得のいく方法をとって下さい。

なお、if-elseでなくswitchを使っているのはやはり見易さと、後でオプションを追加するのが簡単だという理由からです。

```
/*    cat    */

/*    usage: cat <opt> { file }
        opt: -n      with line number.
             -f      .form feed after cat.    */

#include <stdio.h>

#define YES 1
#define NO  0

char line[BUFSIZ];
int  number = NO;
int  formfeed = NO;


main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();
    char *p;

    while ( --argc && *(p=*++argv) == '-' )
        switch ( p[1] ) {
        case 'n':
        case 'N':
            number = YES;
            break;
        case 'f':
        case 'F':
            formfeed = YES;
            break;
        default:
            usage();
            return;
        }
    if ( argc == 0 )
        cat( stdin );
    else {
        if ( ( fp = fopen( p, "r" ) ) == NULL ) {
            fprintf( stderr, "can't open ¥"%s¥".¥n", p );
            return;
        }
        cat( fp );
        fclose( fp );
    }
}


usage()
{
    fprintf( stderr, "usage: cat <-n -f> {file}.¥n" );
}


cat( fp )
    FILE *fp;
{
    int  n;

    for ( n = 1; fgets( line, BUFSIZ, fp ); n++ ) {
        if ( number )
            printf( "%05d ", n );
        fputs( line, stdout );
    }
    if ( formfeed )
        putchar( 0x0c );
}
```

オプションの置き場所についてのもう一つの考え方は、どこに置いてもよいとするものです。これだとまず１回パラメータを全部走査してオプションを評価してから、もう１回走査してオプションでないファイル名を処理しなければなりませんから、結果的には合計２回の走査が必要です。

それをすべてmainで行うのも考えものですから、最初にオプションを評価する部分だけを別の関数optionとします。これをmainから分離すると後でオプションを変更あるいは追加する際に分かり易くなるでしょう。

ついでに関数optionは、パラメータの内でオプション以外のものつまりファイル名が１個でもあるかないかを返すことにします。もう１つついでに、もしオプションに間違いがあった場合はＥＲＲを返すことにします。したがってoptionの返す値はＹＥＳまたはＮＯまたはＥＲＲのどれかです。mainはこの３つの場合についてif-elseで処理を変えます。

optionにはargcとargvをそのまま渡します。これを使って全パラメータを走査するというのがwhile（−−argc）のループです。その内でオプションフラグの'−'で始まるものについてのみswitchを適用します。

パラメータは大文字でも小文字でもそのまま渡されますから、'n'と'N'の両方を認めたい場合は両方を調べてやらなければなりませんので注意して下さい。一部のコマンドに大文字のオプションを認識しないものがあるのはそのためです。

```
/*    cat    */

/*    usage: cat <opt> { file } <opt>
        opt: -n      with line number.
             -f      form feed after cat.     */

#include <stdio.h>

#define YES 1
#define NO  0
#define ERR (-1)

char line[BUFSIZ];
int  number = NO;
int  formfeed = NO;


main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();
    register char *p;
    int  exist;

    if ( ( exist = option( argc, argv ) ) == ERR )
        usage();
    else if ( exist )
        while ( --argc ) {
            if ( *(p=*++argv) == '-' )
                continue;
            if ( ( fp = fopen( p, "r" ) ) == NULL ) {
                fprintf( stderr, "can't open ¥"%s¥".¥n", p );
                return;
            }
            cat( fp );
            fclose( fp );
        }
    else
       cat( stdin );
}



usage()
{
    fprintf( stderr, "usage: <opt> {file} <opt>.¥n" );
    fprintf( stderr, "  opt: -n    with line number.¥n" );
    fprintf( stderr, "       -f    form feed after cat.¥n" );
}


int option( argc, argv )
    int  argc;
    char **argv;
{
    char *p;
    int  exist = NO;

    while ( --argc )
        if ( *(p=*++argv) == '-' ) {
            switch ( p[1] ) {
            case 'n':
            case 'N':
                number = YES;
                break;
            case 'f':
            case 'F':
                formfeed = YES;
                break;
            default:
                return( ERR );
            }
        } else
            exist = YES;
    return( exist );
}


cat( fp )
    FILE *fp;
{
    int  n;

    for ( n = 1; fgets( line, BUFSIZ, fp ); n++ ) {
        if ( number )
            printf( "%05d ", n );
        fputs( line, stdout );
    }
    if ( formfeed )
        putchar( 0x0c );
}
```

以上でオプションフラグは十分ですが、さらにオプションの書き方に柔軟性を持たせるために、'－ｎｆ'のようなものも許すことにしましょう。

このためには１つのオプションの文字列について１文字ずつ見ていかなければなりません。文字列の終端はＣの場合は$00、Basic09の場合は$FF ですからそれが見つかるまでループすることになります。

```
/*    cat    */

/*    usage: cat <opt> { file } <opt>
        opt: -n      with line number.
             -f      form feed after cat.

#include <stdio.h>

#define YES 1
#define NO  0
#define ERR (-1)

char line[BUFSIZ];
int  number = NO;
int  formfeed = NO;


main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();
    register char *p;
    int  exist;

    if ( ( exist = option( argc, argv ) ) == ERR )
        usage();
    else if ( exist )
        while ( --argc ) {
            if ( *(p=*++argv) == '-' )
```

```
                continue;
            if ( ( fp = fopen( p, "r" ) ) == NULL ) {
                fprintf( stderr, "can't open ¥"%s¥".¥n", p );
                return;
            }
            cat( fp );
            fclose( fp );
        }
    else
       cat( stdin );
}


usage()
{
    fprintf( stderr, "usage: <opt> {file} <opt>.¥n" );
    fprintf( stderr, "  opt: -n    with line number.¥n" );
    fprintf( stderr, "       -f    form feed after cat.¥n" );
}

int option( argc, argv )
    int  argc;
    char **argv;
{
    char *p;
    int  exist = NO;
    int  c;

    while ( --argc )
        if ( *(p=*++argv) == '-' ) {
            while ( c = *++p )
                switch ( c ) {
                case 'n':
                case 'N':
                    number = YES;
                    break;
                case 'f':
                case 'F':
                    formfeed = YES;
                    break;
                default:
                    return( ERR );
                }
        } else
            exist = YES;
    return( exist );
}


cat( fp )
    FILE *fp;
{
    int  n;

    for ( n = 1; fgets( line, BUFSIZ, fp ); n++ ) {
        if ( number )
            printf( "%05d ", n );
        fputs( line, stdout );
    }
    if ( formfeed )
        putchar( 0x0c );
}
```

# 1－3　コマンドの組み合わせ

## 1－3－1　パイプラインの応用

ＵＮＩＸでは、すでに作成済みのプログラムを組み合わせて新しいプログラムを作ることを良しとし、Ｃ（など）を新たにコーディングするのを好みません。ＵＮＩＸではShellが一種のインタープリタになっており、それを使えば簡単に新しいコマンドが書けてしまうのです。ＯＳ－９ではShellに制約があるため、限定された範囲でしか実現できません。

許されているのは、’；’や’&’や’！’を使うことだけです。

パイプラインを活用するためには基本的なフィルタ群が完備していなければならず、本書はそのようなフィルタを作ることを目的としています。

フィルタをどう使ったらよいか考えてみましょう。

```
OS9：chd  /d0/defs dsave  !  chd  /d1/defs  shell  －p  t  x
```

dsaveはファイルをコピーするためのプロシジャファイルを出力します。したがってdsaveの出力をディスク上にファイルとしてとっておかなくても、そのままShellに渡してしまえば中間ファイルを作らなくてもよくなります。この場合、chdがShell内蔵コマンドだということが効いています。

さらに、Copyがすでにメモリ上に存在する場合、dsaveが出力する行の中にある’Load copy’と’Unlink copy’を通さないために次のようにします。

```
OS9：chd  /d0/defs  dsave  !  grep  －v  copy  !  chd  /d1/defs
                                                  shell  －p  x  t
```

’grep －v copy’は’copy’という単語が無い行のみ出力するというフィルタです。この場合はたまたまdsaveの出力で’Load copy’と’Unlink copy’の’copy’のところだけ小文字で、他は大文字だということを利用しています。

Shellがワイルドカードをサポートしていないため、ワイルドカード用のコマンドwildを作り、フィルタを通してワイルドカードを実現します。

```
OS9：wild  *  !  place  del  *  !  shell  －p  t  x
```

これでカレントディレクトリ内の全ファイルを削除します。placeは標準入力から１行ずつ読み込んでパラメータ中の'*'と置き替えるフィルタです。最後にshellに送り出しているところに注意して下さい。shellもまたフィルタの一種なのです。

shellに対するオプション'−p'はプロンプトを出さない、'ｔ'は入力行をエコーバックする、'ｘ'は何かエラーが起きた時にアボートするという意味です。この'ｘ'を'−ｘ'にするだけで次のような使い方ができるようになります。

もしディレクトリの中にライトパーミッションがオフのものがあっても、そのファイルの削除に失敗してもアボートしないため、ライト可のものすべてを削除することになります。ただdelの後ろにファイル名を並べて起動するのではこうはいきません。ただしこうすると途中でコントロールＥを押しても止めることができなくなりますので気をつけて下さい。

## Ｉ－3－2　グルーピングの応用

一連のコマンドは、括弧でくくることによりグルーピング化することが可能です。グルーピングとはいくつかのコマンドの集まりを、１つのコマンドとして扱うということですが、ようするに括弧の中身をパラメータとして、新たにshellを起動しているにすぎません。例えば、

```
OS9：echo ABC  ;  echo  DEF  ;  echo  GHI  !  lc
ABC
DEF
1
```

という風に括弧でくくらなければ、最初のechoの出力は端末、２番目のechoの出力は端末、最後のechoの出力はlcの入力となっているので最終的には上のような出力が得られます。これを例えば、

```
OS9：(  echo  ABC  ;  echo  DEF  ;  echo  GHI  )  !  lc
 3
```

という具合に括弧でくくると'ＡＢＣ'と'DEF'と'GHI'という３行の出力が lc に送られることとなり、結果として出力'３'が得られます。

```
OS9：(  echo  merge  ;  wild  ＊  )  !  join  !  shell  −p
```

は、これの応用例でカレントディレクトリの全ファイルをマージすることになります。

また、新たに shell が起動されるという事実に注目して、前に挙げた dsave の例の全体を括弧でくくります。

```
OS9：  (  chd  /d0/sourcedir  dsave  !  chd  /d1/destdir
                                     shell  −p  t  x  )
```

こうすると、終了した時点でのカレントディレクトリの位置は変化していないのです。なぜなら chd は shell の内蔵コマンドであって、chd の置かれた shell のディレクトリが変化するのですが、この場合は新しく起動された方の shell に効果が及ぶのにすぎず、自分自身には影響が無いのです。

つまりいったん

```
OS9：shell
```

としてコマンドを起動して、ＥＯＦ（ＥＳＣキー）によってその shell を終了するのとまったく同等なのです。

単に新しい shell が起動されるだけですからデバイスの占有等の効果はありません。

```
OS9：  (  list  file1  ;  list  file2  )  >/p&
```

のようにしても、プリンタにリストしている間に他のコマンドでプリンタへの出力をすれば、出力は混ざってしまいます。

# Cによるプログラム

# 2－1　C概説

## 2－1－1　argcとargv

本書ではC言語の文法等の話はしません。ただ、コマンド作成という観点から見た場合に必要な事項について簡単に説明します。

Cではプログラムは必ずmainという関数から実行が開始されます。mainはただの関数であって、他の関数と違うところはありません。

mainにも引数が渡されます。argcとargvというものがそれです。この2つでコマンドパラメータを表します。つまり、

argc　は引数の個数を、
argv　は各引数へのポインタからなる配列へのポインタを

表します。

アセンブラのレベルではコマンドパラメータは，唯一個の文字列としてプロセスに渡されるわけですが、Cはそれを「空白で区切られた単語の並び」として単語ごとに分解し、argcとargvに情報をセットしてmainに渡すわけです。

例えば、

param1　param2　param3

というパラメータは‘param1’と‘param2’と‘param3’の3つに分解され、それぞれの置かれた位置（アドレス）を配列に並べ、配列の（先頭の）アドレスがargvとなります。argcには個数がセットされます。

一つ注意することは、コマンドパラメータの第1番目としてそのコマンドの名前が常にセットされているということです。つまりコマンドを起動する時にパラメータを与えなかったとしてもargcは1になっています。

Cでは関数に与えられた引数を無視してもよいことになっていますので、コマンドパラメータを必要としないプログラムでは、引数の宣言をしなくても構いません。

## 2―1―2　入出力

Cには標準の組込関数というものはただの一つもありません。たいていの言語に備わっている基本的な入出力命令の類でさえ、あからさまに関数を呼び出すことによって実現しなければなりません。これらの入出力関数は標準ライブラリとして、システムに合わせて作成されていなければならず、標準ライブラリを使用している限りは(そして、システムによって動作が違うような書き方をしなければ)システム間の移植性が保証されます。逆に言えば、そのようにしてあるからこそ移植性が良いわけです。

システムコールのレベルでの入出力にはOS―9の場合パス番号というものが使用されますが、他のOSではまったく別のものを使用している場合がありますので、パス番号を使った入出力は移植性がありません。そのためにCではシステムごとの入出力の仕様を、ある構造体のなかに押し込めてしまい、ユーザレベルには現れてこないようにしています。

構造体はstdio.hの中に定義されており、バッファの管理やリード／ライトの区別などが格納されています。この構造体は、ＦＩＬＥ型という型としてtypedefで定義してありますので、ユーザはその具体的な内容を知る必要はありません。特に必要がなければこれらの機能を使用するべきです。

ただし、アセンブラを組み込んだりするにはある程度中身を知っていないといけません。OS―9システムコールレベルではパス番号という整数が使用されていますが、このパス番号はＦＩＬＥ構造体の中にセットされ、入出力関数の内部で使用されます。

つまりCにおいては入出力は2重構造になっているのです。パス番号などで表される移植性に問題のある部分は、ＦＩＬＥ構造体を用いた関数群によって吸収されているわけです。

## 2－1－3　文字列

コマンド記述の際の問題の一つが文字列の扱いです。Ｃでは基本的に文字列は可変長のものであり、文字列はその先頭を指すポインタ（アドレス）のよって表されます。したがって文字列の長さを知るためには、最後まで見てみなければなりません。文字列の最後は記号'¥0'（ＡＳＣＩＩコードで＄00）が用いられることになっていますが、これは別に強制ではなく、標準関数の多くがそういう風に扱っているというだけのことです。

このために文字列の扱いが非常に簡単になっているのもＣの特徴の一つと言えるでしょう。

## 2－2　プログラム例

### 2－2－1　join

argcとargvを使わない例としてjoinを挙げます。もちろん引数としてファイルを指定できるようにしても結構です。

標準入力から読み込んだ複数行のテキストを1行に接いで標準出力に書き出します。また1行内で空白が2個以上並んでいるものは、ただ1個の空白で置き換えます。

joinのように機能が単純なものは、コマンドとして役に立つのかと疑問に思われるかもしれません。しかし、そもそもjoinはフィルタとして使うために作ったものなのです。例えば、

```
OS9:( echo merge ; wild *.c ) ! join ! shell -p -t x
```

とすれば、カレントディレクトリのCのソースをすべてマージすることになります。

```
#include <stdio.h>
#define YES 1
#define NO  0

main()
{
    int  space, c;

    space = YES;
    while ( ( c = getchar() ) != EOF )
        if ( c != ' ' && c != '¥n' && c != '¥t' && c != '¥l' ) {
            putchar( c );
            space = NO;
        } else if ( !space ) {
            putchar( ' ' );
            space = YES;
        }
}
```

プログラムを簡単に説明します。全体はＥＯＦまで1バイトずつ読み込むという単純なwhileのループから構成されています。spaceという変数は、1つ前に読み込んだ文字が何だったかを覚えておくためのものです。

空白（タブや改行も空白の一種と考えます）が2個並んでいた場合は、2個目

を読み込んだ時にspaceがＹＥＳになっているはずですから何もしません。3個目以降も同様にしてスキップします。

spaceがＮＯ(つまり前が空白でなかった)で、読み込んだ文字が空白であった場合に初めて空白を出力します。

プログラムの中で‘¥l’(小文字のエル)という見慣れないエスケープキャラクタ($0Aの意味)を使っています。OS-9では復改‘¥n’は$0Dを意味しますが、$0Aは$0Dと同じに解釈した方がよいと考えたために入れたものです。

```
OS9:wild *.c
apart.c
tail.c
dmp.c
cat.c
head.c
err.c
uniq.c
grep.c
fedit.c
join.c

OS9:wild *.c | join
apart.c tail.c dmp.c cat.c head.c err.c uniq.c grep.c fedit.c join.c
```

## 2—2—2　sep

joinとは逆に、単語ごとに切り離すフィルタであるsepを作ります。

```
/*    sep  --  separate words    */

#include <stdio.h>
#define YES 1
#define NO  0

main()
{
    int  space, c;

    space = YES;
    while ( ( c = getchar() ) != EOF )
        if ( c != ' ' && c != '¥n' && c != '¥t' && c != '¥l' ) {
            putchar( c );
            space = NO;
        } else if ( !space ) {
            putchar( '¥n' );
            space = YES;
        }
}
```

これを見て気がついたでしょうか。join.cと1行しか違いません。もっと正確には1文字しか違いません。(コメントは別として。)

参考のためにファイル名を置けるように改良してみます。

```
/*    sep  --  separate words    */

/*    usage: sep (file)    */

#include <stdio.h>

#define YES 1
#define NO  0

main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();

    if ( argc == 1 )
        sep( stdin );
    else
        while ( --argc ) {
            if ( ( fp = fopen( *++argv, "r" ) ) == NULL ) {
                fprintf( stderr, "can't open ¥"%s¥".¥n", *argv );
                return;
            }
            sep( fp );
            fclose( fp );
        }
}



sep( fp )
    FILE *fp;
{
    int  space, c;

    space = YES;
    while ( ( c = getc( fp ) ) != EOF )
        if ( c != ' ' && c != '¥n' && c != '¥t' && c != '¥l' ) {
            putchar( c );
            space = NO;
        } else if ( !space ) {
            putchar( '¥n' );
            space = YES;
        }
}
```

## 2－2－3　place

現在のＯＳ－９にワイルドカードの機能が組み込まれていないため、フィルタを使って肩替わりさせようというのがこのコマンドの主旨です。したがって、たいていはワイルドのコマンドおよびShellと一緒に使用します。

パラメータの中に‘＊’というキャラクタがあった場合に、その位置を入力行でそっくり置き換えてしまいます。例えば、次のようにするとカレントディレクトリの全ファイルのアトリビュートが書き込み可能になります。

```
OS9 : wild  ＊ ! place  attr  ＊  w ! shell -p  t
```

このように、Shellにパイプしないと何の結果も得られません。またwildの‘＊’とplaceの‘＊’は意味がまったく違いますので注意して下さい。

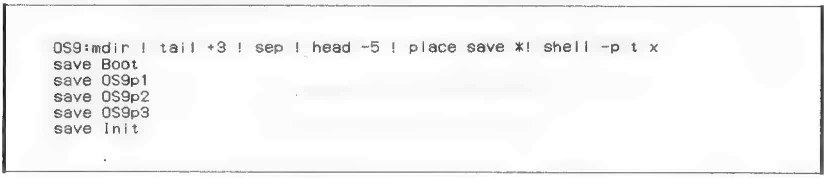
もしShellが機能アップして、ワイルドカードを受け付けるようになった場合は‘＊’でなく、他のキャラクタを使用するようにしなければなりません。

また‘＊’は複数個置いてもよいので、次のような使い方ができます。

```
OS9 : wild  /d0/temp/＊ ! place  copy/d0/temp/＊  ＊! shell  -p  t
```

これで/d0/tempディレクトリにある全ファイルをカレントディレクトリにコピーします。dsaveによく似たコマンドになります。

単なるワイルドカードより融通がきくという例をあげます。



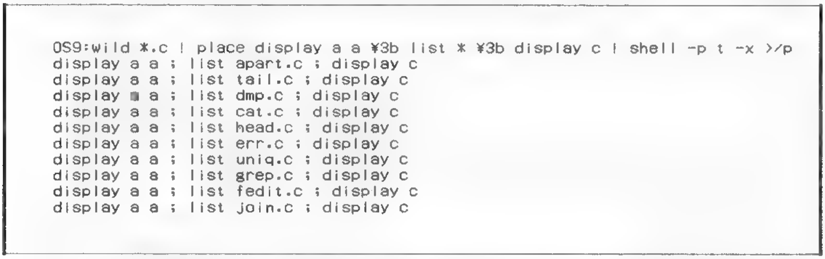
```
OS9:mdir ! tail +3 ! sep ! head -5 ! place save *! shell -p t x
save Boot
save OS9p1
save OS9p2
save OS9p3
save Init
```

これでモジュールディレクトリの最初の５個をカレントディレクトリにsaveしました。headの代わりにgrepを置くなりすれば、モジュールディレクトリに対してワイルドカードを使うこともできます。

文字列の置き換えの他にエスケープキャラクタを用意しました。placeはそもそもShellコマンドラインを出力することを念頭に置いているわけですが、今のままでは次のような出力を得ることはできません。なぜならShellはいかなる場合でも特殊キャラクタを特別なものとして解釈するからです。

list　file1　;　display　c

仕様を次のように決めます。エスケープキャラクタ‘¥’の次の２文字を１６進数と解釈して、その表すキャラクタで置き替えます。例えば、



```
OS9:wild *.c ! place display a a ¥3b list * ¥3b display c ! shell -p t -x >/p
display a a ; list apart.c ; display c
display a a ; list tail.c ; display c
display ■ a ; list dmp.c ; display c
display a a ; list cat.c ; display c
display a a ; list head.c ; display c
display a a ; list err.c ; display c
display a a ; list uniq.c ; display c
display a a ; list grep.c ; display c
display a a ; list fedit.c ; display c
display a a ; list join.c ; display c
```

となり、２行空行の後ファイルをリストして改ページする、というコマンドが出力されているのがわかります。

それでは、プログラムを解説します。

１行ずつ入力し、その文字列(配列line)についてパラメータ全体を適用して文字列を置き換えます。その置き換えそのものは関数putargが行います。しかしCの場合はパラメータが分断されて渡されますので、置き換えを１回ですませることはできません。ここではforループで回して順に置き換え、出力しています。putargする前に１個の空白を出力しないと、パラメータとパラメータの間がくっついてしまいます。forの初期化式のところにputargが入っていてi＝2から始めているのは第１パラメータの前に余計な空白を付けないための処置です。

エスケープキャラクタ‘¥’の処理のために関数ahtoiを使いますが、ahtoiはこれ以降のプログラムでも使用しますし、汎用性のあるものですからDefsディレクトリに格納しておくか、コンパイルして取っておくか、アセンブラに書き直してアセンブルするかしておくと後あと便利だと思います。

```
/*    place  --  replace '*' with STDIN_line    */

#include <stdio.h>
#define ERR (-1)

char line[BUFSIZ];

main( argc, argv )
    int  argc;
    char *argv[];
{
    char *gets();
    int  i;

    if ( argc == 1 )
        return;
    while ( gets( line ) ) {
        if ( *line == '¥0' )
            break;
        for ( putarg( argv[1], line ), i = 2; i < argc; i++ ) {
            putchar( ' ' );
            putarg( argv[i], line );
        }
        putchar( '¥n' );
    }
}


putarg( p, q )          /*  replace '*' in p with q  */
    char *p, *q;
{
    int  c;

    while ( c = *p++ )
        if ( c == '*' )
            fputs( q, stdout );
        else if ( c == '¥¥' && p[0] && p[1] ) {
            putchar( ( ( c = ahtoi(*p)*16+ahtoi(p[1]) ) == ERR )? ' ': c );
            p += 2;
        } else
            putchar( c );
}


int ahtoi( c )
    int  c;
{
    if ( c >= '0' && c <= '9' )
        return( c - '0' );
    else if ( ( c = tolower( c ) ) >= 'a' && c <= 'f' )
        return( c - 'a' + 10 );
    else
        return( ERR );
}
```

## 2－2－4　head

ファイルの最初の方だけを見たい場合に使うのがheadです。listの途中でコントロールEを投入すれば表示を止めることができますが、パイプラインの中にフィルタとして置くためには都合よくありません。headもまたコマンドとしてだけでなくフィルタとして使えるようにしたいのです。

行数は‘－’のあとに１０進数を置くことで指定します。例えば、

OS9　head　－10　samplefile

とすれば、samplefileの先頭の１０行が出力されます。

```
/*    head  ---  print head of file    */

/*    usage: head <opt> {file} <opt>.
        opt: -10  head lines.    */

#include <stdio.h>
#include <ctype.h>

#define YES 1
#define NO  0
#define ERR (-1)

int  count = 10;

main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();
    char *p;
    int  exist;

    if ( ( exist = option( argc, argv ) ) == ERR )
        usage();
    else if ( exist )
        while ( --argc ) {
            if ( *(p=*++argv) == '-' )
                continue;
            if ( fp = fopen( p, "r" ) ) {
                head( fp, count );
                fclose( fp );
            } else {
                fprintf( stderr, "can't open ¥"%s¥".¥n", p );
                return;
            }
        }
    else
        head( stdin, count );
}



usage()
{
    fprintf( stderr, "usage: head <-10> {file} <-10>.¥n" );
}
```

```
int option( argc, argv )
    int  argc;
    char **argv;

{
    register char *p;
    int  exist = NO;

    while ( --argc )
        if ( *(p=*++argv) == '-' )
            if ( isdigit( p[1] ) )
                count = atoi( ++p );
            else
                return( ERR );
        else
            exist = YES;
    return( exist );
}



head( fp, n )
    FILE *fp;
    int  n;
{
    char *fgets();
    char buf[BUFSIZ+1];
    int  i;

    for ( i = 1; i <= n; i++ )
        if ( fgets( buf, BUFSIZ, fp ) )
            fputs( buf, stdout );
        else
            return;
}
```

## 2―2―5　tail

tail は head と逆に、ファイルの終わりの何行かを出力します。

何行であるかを指定するのにオプション'—'と'+'の２つを使用します。'－１０'とするとファイルの終りの１０行という意味になり、'＋３０'とするとファイルの３０行目から終りまでという意味になります。話を簡単にするために、'—'と'+'の両方を置いてはいけないことにします。

ついでに、行番号を付けるオプション'—n'を付加しました。

以下、'—'オプションにまつわる問題点を考えてみましょう。

tail が head の場合と根本的に違うのは、表示する前に、ファイル全体が何行あるかを調べなければならない点にあります。つまり同じファイルを２回読まないと、何行目から出力すべきかがわからないので工夫が必要です。２回読まなければならないということは、１回読んだものを保存しておかなければならないということですが、どこに保存しておくべきかは問題です。ただし、入力ファイルが常にディスクであれば何も問題はありません。１回目は全行数をかぞえ、再び最初から読み直せばよいわけです。それ以外、ＳＣＦタイプのデバイスであったり、パイプラインからの入力だった場合が問題なのです。

以上から、tail では問題をＲＢＦタイプとそうでないものの２つに分けることにしました。ではどうやってＲＢＦタイプかそうでないかを見分けたらよいのでしょう。

マクロ定義の isrbf を見て下さい。Ｃでのファイルは、stdio.h で定義されたＦＩＬＥ構造体配列によって管理されます。その中の _flag という１バイトがファイルのタイプを表しているのです。ただし、オープンしただけではタイプはセットされず、実際にアクセスして初めてセットされます。つまり１バイト読んでみればよいわけですが、その１バイトは ungetc でもどしておかなければなりません。

ただしＦＩＬＥ構造体の中身がどうなっているかは処理系によって異なるものと思わなければならないため、このプログラムがＯＳ－９以外のＯＳのもとで動くかどうかは保証できません。isrbf を書き直すかまたは必ず scftail の方を実行することにすれば問題なくたいていのシステムでも動きます。

scftailでは1回読んだファイルをいったん中間ファイルに格納します。別の考え方として、配列をとってそこにしまっておくという方法がありますが、2つの理由から中間ファイルの方法をとることにしました。

第一に、配列は非常に高くつきます。ファイルの全サイズの分だけ配列をとることはまず不可能です。つまり何行あるかわからない以上はコンパイル時に確保することはできません。できたとしても莫大なメモリを必要とすることになってシングルタスクの環境でならともかく、マルチタスクでは他のプロセスに弊害が出る可能性があります。また動的にメモリを割り当てるために関数mallocを使うことが考えられますが、メモリを大量に消費する可能性がある点ではまったく同じことです。ある一定量まではメモリ上でそれ以上は中間ファイルを使うという手段もあります。試してみる価値はありますが、話が相当に複雑になってしまうでしょう。

第二に、ハードディスクあるいはＲＡＭディスクを使用している環境では、メモリを使っても中間ファイルを使ってもたいした差がありません。特にＲＡＭディスクならばどちらも結局メモリ上にとられますから実行速度もたいして違いません。

さて、以上のような理由により、中間ファイルとしてtail.temp.XXXXXというものを作ります。このXXXXXの位置はmktempによって実際には5桁の整数（実はプロセス番号）に書き換えられます。中間ファイルの削除を行うunlinkは、ようするにdeleteの意味です。

mktempを使わないとどうなるでしょう。中間ファイル名が別々のtailで衝突する可能性が出てきます。例えば1つのパイプラインの中にtailを2つ置いた場合、結果は保証できないことになります。中間ファイルを使う場合はこの点に特に注意して下さい。

```
/*    tail  ---  print tail of file    */

/*    usage: tail <opt> ( file ) <opt>
      opt  : -10     last 10 lines.
             +100    after line 100.
             -n      with line number.    */

#include <stdio.h>

#define LASTLIN 10     /*  default lines  */
#define YES 1
#define NO  0
```

```
#define ERR (-1)
#define isrbf(fp)  ( ungetc(getc(fp),(fp)), (fp)->_flag & _RBF )

char buf[ BUFSIZ ];
int  after = 0;        /*    print after 'after' line    */
int  ntail = 0;        /*    print last 'ntail' lines   */
int  number;           /*    with line number ?         */

main( argc, argv )
    int  argc;
    char *argv[];
{
    register char *p;
    FILE *fp, *fopen();
    int  exist, c;

    if ( ( exist = option( argc, argv ) ) == ERR )
        usage();
    else if ( exist ) {
        while ( --argc )
            if ( ( c =*(p=*++argv) ) != '-' && c != '+' ) {
                if ( ( fp = fopen( p, "r" ) ) == NULL ) {
                    fprintf( stderr, "can't open ¥"%s¥".¥n", p );
                    return;
                }
                if ( isrbf( fp ) )
                    rbftail( fp );
                else
                    scftail( fp );
                fclose( fp );
            }
    } else
        scftail( stdin );
}


usage()
{

{
    fprintf( stderr, "usage: tail <-10 | +99> <-n> {file} <opt>.¥n" );
    fprintf( stderr, "  opt: -n     with line number.¥n" );
    fprintf( stderr, "       -10    last 10 lines.¥n" );
    fprintf( stderr, "       +99    after line 99.¥n" );
}


int option( argc, argv )
    int  argc;
    char **argv;
{
    char *p;
    int  c;
    int  exist = NO;

    while ( --argc )
        if ( ( c = *(p=*++argv) ) == '-' || c == '+' )
            switch ( c ) {
            case '-':
                if ( *++p == 'n' )
                    number = YES;
                else
                    ntail = atoi( p );
                break;
            case '+':
                after = atoi( ++p );
                break;
            default:
                return( ERR );
            }
        else
            exist = YES;
    if ( ntail && after )
        return( ERR );
    else if ( ntail == 0 && after == 0 )
```

```
        ntail = LASTLIN;            /* default lines */
    return( exist );
}


rbftail( fp )
    FILE *fp;
{
    int  c, i, n;

    for ( n = 0; fgets( buf, BUFSIZ, fp ); n++ )
        ;
    rewind( fp );
    putnline( fp, n );
}


scftail( fp )
    FILE *fp;
{
    char tempname[ 32 ];
    FILE *temp, *fopen();
    int  n, i;

    mktemp( strcpy( tempname, "tail.temp.XXXXX" ) );
    if ( ( temp = fopen( tempname, "w" ) ) == NULL ) {
        fprintf( stderr, "can't create temp_file.¥n" );
        return;
    }
    for ( n = 0; fgets( buf, BUFSIZ, fp ); n++ )
        fputs( buf, temp );
    fclose( temp );
    if ( ( temp = fopen( tempname, "r" ) ) == NULL ) {
        fprintf( stderr, "can't open temp_file.¥n" );
        return;
    }
    putnline( temp, n );
    fclose( temp );
    unlink( tempname );    /* delete temporaly file */
}


putnline( fp, n )
    FILE *fp;
    int  n;
{
    int  i;

    n = (ntail)? n-ntail: after-1;    /* count for skip lines */
    for ( i = 1; i <= n; i++ )
        fgets( buf, BUFSIZ, fp );
    for ( ; fgets( buf, BUFSIZ, fp ) != NULL; i++ )
        if ( number )
            printf( "%05d %s", i, buf );
        else
            fputs( buf, stdout );
}
```

## 2－2－6　uniq

uniqは重複する行を取り除いて1行にまとめるコマンド（フィルタ）です。

abcdefg
uvwxyz
uvwxyz
aaa
と入力すると
abcdefg
uvwxyz
aaa
と出力します。

uniqにはオプションがありません。またファイル名を複数個置いてよいことにし、ファイル名を1つも指定しなかった場合は標準入力から読み込みます。

1つのファイルについて重複を取り除く作業をするのが関数uniqです。行が重複しているかどうかを調べるために、バッファを2つ用意しておき、直前に読んだ行をbeforeの方に取っておきます。もし今後読んだ行が前の行（before）と等しければ出力しません。もし等しくなければ出力し、その行をbeforeにコピーします。なお、文字列の比較には標準関数のstrncmpを使います。strncmpがstrcmpと違うのは、比較する文字列の最大サイズを指定できるということですが、指定しても結局のところ意味がありませんのでstrcmpにしても結構です。

第1行目だけは前の行と比較しようにも前の行自体が存在しないわけですが、beforeの初期値を空文字列としておくことによりstrncmpの結果が必ず「等しくない」となりますから必ず出力されます。なお、beforeはグローバル（というよりはスタティック）に宣言されているのでコマンド起動は空文字列になっているのですが、2つ目以降のファイルを調べる時に、前のファイルの最後のbeforeの値が残っているため、結局は関数uniqの最初で初期化しなければなりません。

```
/*    uniq  --  print unique line    */

/*    usage: uniq (file).    */

#include <stdio.h>

#define EQUAL 0

char before[BUFSIZ];
char line[BUFSIZ];

main( argc, argv )
```

```
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();

    if ( argc == 1 )
        uniq( stdin );
    else
        while ( --argc )
            if ( fp = fopen( *++argv, "r" ) ) {
                uniq( fp );
                fclose( fp );
            } else {
                printf( stderr, "can't open ¥"%s¥".¥n", *argv );
                return;
            }
}



uniq( fp )
    FILE *fp;
{
    *before = '¥0';
    while ( fgets( line, BUFSIZ, fp ) )
        if ( strncmp( before, line, BUFSIZ ) != EQUAL ) {
            fputs( line, stdout );
            strcpy( before, line );
        }
}
```

## 2―2―7　apart

OS9Boot ファイルのようにモジュールが複数個マージしてあるファイルをばらばらのファイルにするプログラムです。

システムの変更をする際には OS9Boot を作り直すことが多いですが、そのためにはあらかじめ OS9Boot の中のモジュールを別々のファイルにしておく必要があります。ただしすべてがメモリモジュール上にあるのならば、save を繰り返すという方法もあるにはありますが、特にレベル 2 では、OS9Boot の書き換え以外にも、パッケージ化したコマンドを扱うことが多いので、apart はなかなか有用です。

例えば、筆者のシステムではコマンドをパッケージ化して Startup 時にロードするようにしています。

```
OS9:wild -x *.etc ! head -3
attr.etc
date.etc
exbin.etc
```

このうちattr.etcは次のものをマージしたものです。(これらのファイル名には深い意味はありません。attrが1番最初でetcetraというほどのつもりです。)

```
OS9:mh attr.etc -xe

'attr.etc'
module name      size  type    lang     at/rv
---------------- ----  ------  ------   -- --
Attr             0286  Prgrm   Objct    80 01
Build            0054  Prgrm   Objct    80 01
Del              00A5  Prgrm   Objct    80 01
Dir              0300  Prgrm   Objct    80 01
Display          0084  Prgrm   Objct    80 01
Echo             0022  Prgrm   Objct    80 01
Link             002C  Prgrm   Objct    80 01
Makdir           0022  Prgrm   Objct    80 01
MDir             026A  Prgrm   Objct    80 01
Merge            0068  Prgrm   Objct    80 01
Mfree            01EB  Prgrm   Objct    80 01
Procs            0280  Prgrm   Objct    80 01
Rename           011D  Prgrm   Objct    80 01
Unlink           002D  Prgrm   Objct    80 01
```

このattr.etcにもう一つモジュールを加えるのは容易です。単にマージすればよいのですから。しかし一つ削除するのは簡単ではありません。全部saveし直すかまたはapartを使います。

まず新しいディレクトリを作った方がよいでしょう。そこに移った後、

```
OS9 : apart   /d0/cmds/attr.etc
```

で、カレントディレクトリに１４個のファイルができます。これらを削除するなり入れ替えるなりした後、

```
OS9:(echo merge;wild *)!join! shell -p>/d0/cmds/ attr.etc
```

などとすれば出来上がりです。なお、このあとパーミッションを‘e’（実行可能）にしておかないとロードできません。

さて、オプションとして‘−x’と‘−d’の２つを付けます。‘−x’はもとのファイルがカレント実行ディレクトリにあることを表します。

もう一つの‘−d’オプションで目的のファイルを作る場所を任意のディレクトリに設定できるようにしました。例えば次の場合を考えます。

```
OS9 : apart   /d0/cmds/attr.etc   /d0/cmds/date.etc
```

これは/d0/cmdsというディレクトリにある2つのファイルをカレントディレクトリに分解するものですが、完全なパスリストを2つとも与えなければならないわけで、打ち込むのが面倒です。むしろ/d0/cmdsに移っておいて、格納先のディレクトリの方を指定した方が簡単です。

```
OS9 : chd /d0/cmds apart attr.etc date.etc −d=/dl/temp
```

ではプログラムを説明します。

mainはoptionを呼び出してオプションを設定します。−dで指定したディレクトリ名はrootにコピーしておきます。−dを指定しないとrootは空のままです。

関数apartは1つのファイルを分解します。mainはそれぞれのファイルごとにapartを呼び出します。apartはまずヘッダを読み込んでモジュールの正当性をチェックし、中間ファイルをクリエイトします。そしてモジュール1個をコピーした後中間ファイルを本来のモジュール名にrenameします。

最初からモジュールの名前でクリエイトしないで、renameするというのは二度手間のように思われるかもしれませんが、これには理由があります。モジュールの名前はモジュール本体の不定位置にあるため、クリエイトすべき時点ではまだ名前がわからないのです。いったん名前を読み込んだ後でfseekでもどるという手もありますが、入力ファイルとしてディスク以外のものを許しているためにできません。したがっていったん中間ファイルにコピーした後でその中をfseekしてモジュール名を求めます。“w+”でfopenしているのはそのためです。

さて一つだけ妥協をします。モジュールの正統性のチェックはシンクバイトのみについてだけ行い、ヘッダパリティおよびCRCは調べません。たいていの場合はそれだけで間に合うはずですし、不都合が起きたとしても我慢しましょう。どうしても気になる方は、たいした手間ではありませんから自分で組み込んでみて下さい。

```
/*    apart packaged module file    */

/*    usage: apart <opt> (file) <opt>.
        opt: -x           source is in execution directory.
             -d=<path>  destination directory.    */

#include <stdio.h>
```

```
#define YES 1
#define NO  0
#define ERR (-1)

char root[BUFSIZ], pathlist[BUFSIZ];
int  exec = NO;

main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen(), *fclose();
    register char *p;
    int  exist;

    if ( ( exist = option( argc, argv ) ) == ERR )
        usage();
    else if ( exist )
        while ( --argc ) {
            if ( *(p=*++argv) == '-' )
                continue;
            if ( fp = ( exec? fopen( p, "rx" ): fopen( p, "r" ) ) ) {
                if ( apart( fp ) == ERR ) {
                    fclose( fp );
                    break;
                } else
                    fclose( fp );
            } else {
                fprintf( stderr, "can't open ¥"%s¥".¥n", p );
                return;
            }
        }
    else
        apart( stdin );
}


usage()
{
    fprintf( stderr, "usage: <opt> {file} <opt>.¥n" );
    fprintf( stderr, "  opt: -x           execution directory.¥n" );
    fprintf( stderr, "       -d=pathlist  destination directory.¥n" );
}

int option( argc, argv )
    int argc;
    char **argv;
{
    register char *p;
    int  exist = NO;

    while ( --argc )
        if ( *(p=*++argv) == '-' ) {
            switch ( *++p ) {
            case 'x':
            case 'X':
                exec = YES;
                break;
            case 'd':
            case 'D':
                if ( *++p == '=' )
                    strcpy( root, ++p );
                else
                    return( ERR );
                break;
            default:
                return( ERR );
            }
        } else
            exist = YES;
    return( exist );
}


int apart( fp1 )
```

```
    FILE *fp1;
{
    FILE *fp2, *fopen(), *fclose();
    struct {
        unsigned sync, size, name;
        char tylan, atrev, hpar;
    }, head;
    char fname[32];

    while ( fread( &head, sizeof(head), 1, fp1 ) ) {
        if ( head.sync != 0x87cd ) {
            fprintf( stderr, "header is incorrect.¥n" );
            return( ERR );
        }
        mktemp( strcpy( fname, "apart.tempXXXXX" ) );
        makepath( pathlist, root, fname );
        if ( ( fp2 = fopen( pathlist, "w+" ) ) == NULL ) {
            fprintf( stderr, "can't create. ¥"%s¥".¥n", pathlist );
            return( ERR );
        }
        fwrite( &head, sizeof(head), 1, fp2 );
        if ( filecopy( fp1, fp2, (long)(head.size-sizeof(head)) ) == ERR ) {
            fprintf( stderr, "end of file.¥n" );
            fclose( fp2 );
            return( ERR );
        }
        fseek( fp2, (long)head.name, 0 );
        makename( fp2, fname );
        rename( pathlist, fname );
        fclose( fp2 );
    }
    return( NO );
}

makename( fp, p )
    FILE *fp;
    register char *p;
{
    for ( ; ( ( *p = getc( fp ) ) & 0x80 ) == 0; p++ )
        ;
    *p &= 0x7f;
    *++p = '¥0';
}


makepath( pathlist, root, fname )
    char *pathlist, *root, *fname;
{
    if ( *root ) {
        strcpy( pathlist, root );
        strcat( pathlist, "/" );
        strcat( pathlist, fname );
    } else
        strcpy( pathlist, fname );
}


int filecopy( fp1, fp2, n )
    FILE *fp1, *fp2;
    long n;
{
    register int  c;

    while ( n-- )
        if ( ( c = getc( fp1 ) ) == EOF )
            return( ERR );
        else
            putc( c, fp2 );
    return( NO );
}


rename( p, q )
    char *p, *q;
```

```
{
    char line[81];

    strcpy( line, "rename " );
    strcat( line, p );
    strcat( line, " " );
    strcat( line, q );
    system( line );
}
```

## 2―2―8　dmp

ファイルの１６進ダンプのプログラムです。ファイルのアドレスは6桁で表示されます。オプションによってスタートアドレスとエンドアドレスを指定でき、省略時はファイル全部をダンプします。

指定したファイルがディレクトリだった場合でも普通のファイルとまったく同じに扱います。つまりディレクトリをダンプするのに特別なオプションを使わずに、ファイルの種類を自動的に判別します。

関数dmpが一つのファイルについてダンプを実行します。一般にファイルアドレスは３２ビットのロングインテジャにしなければなりませんのでリスト中ではlong宣言をしています。

dmpはスタートアドレスsadrからエンドアドレスeadrまでループします。スタートアドレスまでスキップする関数skipでは、tailの場合と同様にＲＢＦタイプならばfseekしますが、それ以外は1バイトずつ読み込みます。

さて、表示は１６バイト単位で行いますが、スタートアドレスとエンドアドレスの間にないものは1バイトたりとも表示しないようにするため、表示用のバッファbufを設けてそこには1バイトずつ書き込んでいき、一杯になったら出力するという方法を使います。ようするにオープンやクローズと同じことをbufについて行うわけです。

dmpの中のinitbufがオープンで、flashbufがクローズに担当すると思えばよいでしょう。変数keepはバッファにデータがあるかないかを表します。これはデータが無いのに書き出してしまうのを避けるためです。

addbufがputcに担当します。addbufは１６進2桁の部分とＡＳＣＩＩコードの部分をbufにセットし、xxxx00のアドレスだったならばヘッダを出力します。

また xxxxxf のアドレスだったならば buf を出力 (flashbuf) し、buf をもとにもどします (initbuf)。

initbuf は buf をスペースで埋めますので１６バイトのうちデータが無い場所はスペースが出力されることになります。

```
/*    hex-decimal dump     */

/*    usage: dmp <opt> {file} <opt>     */

/*    option: -x            execution directory.
              -s=000000     start address.
              -e=ffffff     end   address.     */

#include <stdio.h>

#define isrbf(fp)  ( ungetc(getc(fp),(fp)), (fp)->_flag&_RBF )

#define ERR (-1)
#define YES 1
#define NO  0
#define HEXOFT  8
#define CHROFT 49
#define BUFSIZ (6+2+(4+1)*8+1+16)

long start = 0;              /*  start offset   */
long endadr = 0x7fffffff;    /*  end address    */
char buf[BUFSIZ+1];
int  keep = NO;
int  exec = NO;


main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *dopen();
    int  exist;
    register char *p;

    if ( ( exist = option( argc, argv ) ) == ERR )
        usage();
    else if ( exist ) {
        while ( --argc ) {
            if ( *(p=*++argv) == '-' )
                continue;
            if ( ( fp = dopen( p, exec ) ) == NULL ) {
                fprintf( stderr, "can't open ¥"%s¥".¥n", p );
                return;
            } else {
                dmp( fp, start, endadr );
                fclose( fp );
            }
        }
    } else
        dmp( stdin, start, endadr );
}



FILE *dopen( p, exec )
    register char *p;
    int  exec;
{
    FILE *fp, *fopen();

    if ( exec )
        return( ( (fp=fopen(p,"rx")) || (fp=fopen(p,"dx")) )? fp: NULL );
    else
        return( ( (fp=fopen(p,"r")) || (fp=fopen(p,"d")) )? fp: NULL );
}


usage()
{
    fprintf( stderr, "usage: dmp <opt> (file) <opt>.¥n" );
    fprintf( stderr, "  opt: -x          execution directory.¥n" );
    fprintf( stderr, "       -s=000000   start offset.¥n" );
    fprintf( stderr, "       -e=ffffff   end   offset.¥n" );
}



int option( argc, argv )
    int  argc;
    char **argv;
{
    long htol();
    register char *p;
    int  exist = NO;

    while ( --argc )
        if ( *(p=*++argv) == '-' )
            switch( *++p ) {
            case 'x':
            case 'X':
                exec = YES;
            case 's':
            case 'S':
                ++p;
                start = htol( ( *p == '=' )? ++p: p );
                break;
            case 'e':
            case 'E':
                ++p;
                endadr = htol( ( *p == '=' )? ++p: p );
                break;
            default:
                return( ERR );
            }
        else
            exist = YES;
    return( exist );
}



dmp( fp, sadr, eadr )
    FILE *fp;
    long sadr, eadr;
{
    register int  c;

    if ( sadr && skip( fp, sadr ) == EOF )
        return;
    initbuf( (long)( sadr & ~0x0f ) );
    if ( ( sadr & 0xff ) != 0 )
        header();
    for ( keep = NO; sadr <= eadr && ( c = getc( fp ) ) != EOF; sadr++ )
        addbuf( sadr, c );
    flashbuf();
}



int skip( fp, adr )
    FILE *fp;
    long adr;
{
    if ( isrbf( fp ) )
        fseek( fp, adr, 0 );
    else
        while ( adr-- )
            if ( getc( fp ) == EOF )
                return( EOF );
    return( NO );
}
```

```
header()
{
    putchar( '¥n' );
    puts(" Addr    0 1  2 3  4 5  6 7  8 9  a b  c d  e f  0 2 4 6 8 a c e");
    puts("------  ---- ---- ---- ---- ---- ---- ---- ----  ----------------");
}



initbuf( adr )
    long adr;
{
    register int  i;

    itoh6( buf, adr );
    for ( i = 6; i < BUFSIZ; i++ )
        buf[i] = ' ';
    buf[BUFSIZ] = '¥0';
}



flashbuf()
{
    if ( keep )
        puts( buf );
}



addbuf( adr, c )
    long adr;
    int  c;
{
    register int  n;

    keep = YES;
    n = adr & 0x0f;
    buf[ CHROFT + n ] = toprint( c & 0xff );
    itoh2( buf + HEXOFT + (n>>1)*5 + ((n&1)?2:0) , c );
    if ( ( adr & 0xff ) == 0 )
        header();
    else if ( n == 15 ) {
        flashbuf();
        initbuf( (long)(adr+1) );
        keep = NO;
    }
}



int toprint( c )
    register int  c;
{
    return( (c<=0x1f||c==0x7f||c==0xa0||c==0x7e||c==0xff)? '.': c );
}



long htol( p )
   char *p;
{
    long val = 0L;
    int  c;

    while ( c = *p++ )
        if ( c >= '0' && c <= '9' )
            val = ( (val<<4) & ~0x0f ) + ( c - '0' );
        else if ( ( c |= 0x20 ) >= 'a' && c <= 'f' )
            val = ( (val<<4) & ~0x0f ) + ( c - 'a' + 10 );
        else
            break;
    return( val );
}
```

```
itoh6( p, val )
    register char *p;
    long val;
{
    int  i = 6;
    int  c;

    p += 6;
    while ( i-- ) {
        c = val & 0x0f;
        *--p = itoh( c );
        val >>= 4;
    }
}



int itoh( c )
    register int  c;
{
    return( ( c >= 0 && c <= 9 )? c+'0': c+'a'-10 );
}



itoh2( p, c )
    register char *p;
    int  c;
{
    *p = itoh( (c)>>4) & 0x0f );
    p[1] = itoh( c & 0x0f );
}
```

```
 Addr    0 1  2 3  4 5  6 7  8 9  a b  c d  e f   0 2 4 6 8 a c e
------   ---- ---- ---- ---- ---- ---- ---- ----  ----------------
000040   4c69 6eeb 0000 0000 0000 0000 0000 0000  Lin♠............
000050   0000 0000 0000 0000 0000 0000 0000 0317  ................
000060   4d61 6b44 69f2 0000 0000 0000 0000 0000  MakDi年..........
000070   0000 0000 0000 0000 0000 0000 0000 0319  ................
000080   4d64 69f2 0000 0000 0000 0000 0000 0000  Mdi年............
000090   0000 0000 0000 0000 0000 0000 0000 031b  ................
0000a0   4d65 7267 e500 0000 0000 0000 0000 0000  Merg◣...........
0000b0   0000 0000 0000 0000 0000 0000 0000 031f  ................
0000c0   4d66 7265 e500 0000 0000 0000 0000 0000  Mfre◣...........
0000d0   0000 0000 0000 0000 0000 0000 0000 0321  ...............!
0000e0   5072 6f63 f300 0000 0000 0000 0000 0000  Proc月...........
0000f0   0000 0000 0000 0000 0000 0000 0000 0324  ...............$

 Addr    0 1  2 3  4 5  6 7  8 9  a b  c d  e f   0 2 4 6 8 a c e
------   ---- ---- ---- ---- ---- ---- ---- ----  ----------------
000100   5265 6e61 6de5 0000                      Renam◣..
```

## 2—2—9　fold

foldはファイルを指定した桁数で折りたたむコマンドです。桁数はheadの時と同様に‘−20’のようにして指定します。コマンドとしてもフィルタとしても使用することができるのはいつもの通りです。

プログラムもまたheadとほとんど同じですので、関数foldについてだけ説明します。foldはfpによって指定されたファイルから1文字ずつ読み込んでいきます。桁を数えるために変数nを用意しておき、1文字読み込むたびにインクリメントしていきますが、rangeに達するかまたは行の終りに達した時に改行してnを0にもどします。

foldは次のようにして起動します。

```
OS9 : fold　fold.c　−10 ! head　−10
```

```
/*    fol
d    */

/*    usa
ge: fold <
-80> {file
} <-80>.
  */

#include
```

```
/*    fold    */

/*    usage: fold <-80> {file} <-80>.    */

#include <stdio.h>
#include <ctype.h>

#define YES 1
#define NO  0
#define ERR (-1)

int  range = 80;    /*  range of line  */

main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();
    char *p;
    int  exist;

    if ( ( exist = option( argc, argv ) ) == ERR || range == 0 ) {
        usage();
        return;
    } else if ( exist )
        while ( --argc ) {
            if ( *(p=*++argv) == '-' )
                continue;
            if ( ( fp = fopen( p, "r" ) ) == NULL ) {
                fprintf( stderr, "can't open ¥"%s¥".¥n", p );
                return;
            }
            fold( fp );
            fclose( fp );
        }
    else
        fold( stdin );
}


usage()
{
    fprintf( stderr, "usage: fold <-80> {file} <-80>.¥n" );
}


int option( argc, argv )
    int  argc;
    char **argv;
{
    char *p;
    int  exist = NO;
    while ( --argc )
        if ( *(p=*++argv) == '-' ) {
            if ( isdigit( *++p ) )
                range = atoi( p );
            else
                return( ERR );
        } else
            exist = YES;
    return( exist );
}


fold( fp )
    FILE *fp;
{
    int  c, n;

    n = 0;
    while ( ( c = getc( fp ) ) != EOF ) {
        if ( c == '¥n' )
            n = 0;
        else if ( ++n >= range ) {
            putchar( '¥n' );
            n = 0;
        }
        putchar( c );
    }
}
```

## 2—2—10 fed

このコマンドはファイルをスクリーンエディットするものです。

読み書きは１セクタを単位として行われます。１セクタのエディットが終わるまで実際にそのセクタが書き換えられることはありません。

起動の仕方は、

```
OS9 : fed   file
```

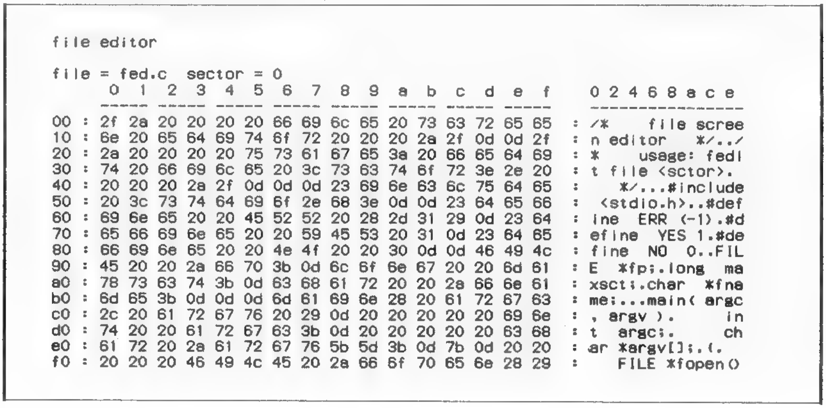
のようにファイル名を指定します。また、ファイルの途中から始めたい場合は

```
OS9 : fed   file   10
```

と、ファイル位置を２５６バイト単位で測った時のセクタ番号で（１０進数）指定します。ファイル名とセクタ番号を置く位置は交換できません。

この場合のセクタとは、単にファイル位置を２５６バイト単位で数えた値にすぎず、ディスク管理上の論理セクタとは関係ありません。ただしファイル名として‘@’付きのものを指定すれば論理セクタと同義になります。

なお、ディレクトリファイルをエディットすることは今のところ安全性のために禁止してあります。



```
file editor

file = fed.c  sector = 0
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0 2 4 6 8 a c e
     ----- ----- ----- ----- ----- ----- ----- -----    ----------------
00 : 2f 2a 20 20 20 20 66 69 6c 65 20 73 63 72 65 65  : /*    file scree
10 : 6e 20 65 64 69 74 6f 72 20 20 20 2a 2f 0d 0d 2f  : n editor   */../
20 : 2a 20 20 20 20 75 73 61 67 65 3a 20 66 65 64 69  : *    usage: fedi
30 : 74 20 66 69 6c 65 20 3c 73 63 74 6f 72 3e 2e 20  : t file <sctor>. 
40 : 20 20 20 2a 2f 0d 0d 0d 23 69 6e 63 6c 75 64 65  :    */...#include
50 : 20 3c 73 74 64 69 6f 2e 68 3e 0d 0d 23 64 65 66  :  <stdio.h>..#def
60 : 69 6e 65 20 20 45 52 52 20 28 2d 31 29 0d 23 64  : ine  ERR (-1).#d
70 : 65 66 69 6e 65 20 20 59 45 53 20 31 0d 23 64 65  : efine  YES 1.#de
80 : 66 69 6e 65 20 20 4e 4f 20 20 30 0d 0d 46 49 4c  : fine  NO  0..FIL
90 : 45 20 20 2a 66 70 3b 0d 6c 6f 6e 67 20 20 6d 61  : E  *fp;.long  ma
a0 : 78 73 63 74 3b 0d 63 68 61 72 20 20 2a 66 6e 61  : xsct;.char  *fna
b0 : 6d 65 3b 0d 0d 0d 6d 61 69 6e 28 20 61 72 67 63  : me;...main( argc
c0 : 2c 20 61 72 67 76 20 29 0d 20 20 20 20 20 69 6e  : , argv ).     in
d0 : 74 20 20 61 72 67 63 3b 0d 20 20 20 20 20 63 68  : t  argc;.     ch
e0 : 61 72 20 2a 61 72 67 76 5b 5d 3b 0d 7b 0d 20 20  : ar *argv[];.{.  
f0 : 20 20 20 46 49 4c 45 20 2a 66 6f 70 65 6e 28 29  :    FILE *fopen()
```

起動しますとファイル名、セクタ番号および１セクタ分のデータがスクリーン上に表示され、カーソルが点滅します。カーソルキーを使って任意の位置に移動してエディットして下さい。

ＨＥＸモードとは１６進２桁による入力モードで、ＡＳＣＩＩモードとはキャラクタによる入力モードです。

次のキーは特別な意味を持ちます。

| | |
|---|---|
| →　または　SPACE | １文字右移動 |
| ←　または　BS | １文字左移動 |
| ↑ | １文字上移動 |
| ↓ | １文字下移動 |
| SHIFT ＋ → またはCNTL＋F | ASCII モードへ |
| SHIFT ＋ ← またはCNTL＋B | HEX モードへ |
| SHIFT ＋ ↑ またはCNTL＋Y | 前のセクタへ |
| SHIFT ＋ ↓ またはCNTL＋Z | 次のセクタへ |
| HOME | ホームポジションへ |
| CR | 次の行の先頭へ |

※ただし FM-11 では SHIFT＋カーソルキーは使えません。

プログラム全体は大きく３つの部分に分けることができます。そのためソースを３つに分けて、分割コンパイルをしています。

## １．　メインプログラム

argc および argv をもとに、エディットするファイルをオープンし、最初にエディットするセクタを求めます。ファイルのエディットは関数 edit が行いますが、その前後でエコーバックの禁止および許可を行います。スクリーンエディットの際には１文字入力１文字出力をもとにしていますが、コントロールキー等をそのままエコーバックされては困りますので edit の前に強制的にエコーバックを禁止し、プログラム終了時にエコーバックをもとにもどします。

また、入出力をＣの方でバッファリングされるとリアルタイムの応答ができなくなってしまいますので、あらかじめ setbuf によってアンバッファリングのモードにしておきます。

ファイルの長さ以上のエディットを禁止するために、変数 maxsct にファイルの最大セクタ番号をセットします。これは他の関数からも参照しますし、ほぼ定数のような意味あいを持っていますからグローバルに宣言しておきます。

エコーバックの制御方法として最も簡単なのは system 関数を使って、

```
system( “ tmode  －echo” )；
```

とすることですが、いちいち tmode を起動するのも面倒ですから、システムコールを使って直接パスディスクリプタを書き換えてしまいます。ＯＳ－９のＣにはそのための関数 getstat および setstat が用意されています。

### ２． edit

セクタ移動とファイルの実際の入力および出力、バッファの管理をします。

edit への引数 sct はそのセクタから始めることを表しますが、エデットするセクタは edit の中でどんどん動きます。エディットした結果をキー入力の度にディスクに書き込むわけではなく、いったん２５６バイトのバッファ上で修正しておいて、そのセクタを抜ける時に実際にセクタごと書き込みます。なお抜ける際には必ず書き込んでよいかどうかの確認をとりますので、‘y’か‘n’で応答して下さい。

edit はまず指定されたセクタの内容をバッファ buf に読み込みますが、エラーだった場合は画面の下の方に“fed　abort.”を表示して終了します。

正常に読み込んだ場合は chgbuf を呼び出します。この chgbuf がスクリーンエディットによってバッファの内容を書き換えます。chgbuf はスクリーンエディットを抜けた時のキーコードを返します。それは次の３種類に限定されています。

| | |
|---|---|
| EOF | 無条件にプログラムを終了 |
| BACK | １セクタ前にもどる。 |
| NEXT | １セクタ先へ進む。 |

このうち、chgbufの返した値がＥＯＦだった場合はwhileループを抜けて、“fed　abort.”を表示して終了します。この場合はそのセクタへの書き込みは行いません。

ＥＯＦを無条件終了としたのには理由があります。スクリーンエディットは標準入力からのキー入力をもとにしているため、キーボードのＥＯＦであるＥＳＣが押されるとそれ以降の入力ができなくなってしまうのです。ＥＳＣを無効にしておいてもよいのですが、プログラムの終了キーとして手頃であったのと、入力をリダイレクトしても使えることになるということから、これをそのままにしておくことにしました。(リダイレクトは危険ですからやらない方がよいでしょう。)

ＢＡＣＫおよびＮＥＸＴだった時、１セクタ移動した先がファイルの範囲にあるかどうかを調べ、もしマイナスだったりファイルのサイズを越えていた場合は同じセクタをもう一度やり直します。

セクタ読み込みはreadbufで行います。セクタ番号をもとにしてしかるべき位置へシークし、２５６バイトをgetcします。途中でＥＯＦが検出された場合はそれ以後いくら読み込んでもＥＯＦが返され続けますから、ＥＯＦ検出時にはゼロを入力したことにしてバッファにはゼロを代入しておきます。

セクタへの書き込みはwritebufで行います。readbufの時と同様に１文字ずつputcで書き込んでいきますが、ファイルの終りに達した場合はＥＯＦが返されますので、その時breakして書き込みをやめます。

したがって、ファイルの最後のセクタをエディットした場合、EOFから先のバイトもエディットそのものはできますが、書き込まれませんので注意して下さい。

書き込む前にwriteokで確認をとります。もし何か間違えてしまってそのセクタを書き込んでほしくないときは‘n’と押して下さい。一瞬ですが、

```
writing..
```

または、

Not　write..

のメッセージを表示して次の（または前の）セクタに移ります。

スクリーンエディットに移る前に、読み込んだバッファの内容を画面に表示するのが関数dumpです。これは、少し書き換えるだけで普段のデバッグ時に使える関数になりますので、後でcoredump関数として示します。

### ３．　chgbuf

スクリーンエディットをするのが関数chgbufです。入力にはＨＥＸモードとＡＳＣＩＩモードの２つがあることは前に述べましたが、それらのモードの間の移動を制御するのを主な役割とします。

実際にはＨＥＸモードでのエディットはhexchangeで行い、ＡＳＣＩＩモードでのエディットはascchangeで行います。これらの関数はモードの移動キーまたはエディットから抜けるキーが押された時にそのキーを返しますので、それがモード移動以外だった場合に無限ループを抜けます。

ＨＥＸモードでは、１６進２桁のうち上位バイトにいるのか下位バイトにいるのか覚えていなければならないので、上位バイトの時ＹＥＳ、下位バイトの時にＮＯの値を持つhighをフラグとして使用します。スクリーンエディット時のキーの動きはこのhighの値によって変わってきます。

キー入力とそれに対応する動きは大きなcase文で場合分けしてあります。この中で使っているhexlocateは、ＨＥＸモード時にバッファ中でのポインタptとhighによってしかるべき位置にカーソルを移動させる関数です。また、モードの如何を問わず、データを書き変えた時は１６進とＡＳＣＩＩの両方を表示し直さなければなりません。関数putbufはそのためのもので、ＡＳＣＩＩモードからも呼ばれます。

ＡＳＣＩＩモードでは、ptの値によって一位的に座標が決定されますので話はいくらか簡単になります。やはり大きなcase文によってキーと動きを対応付けます。

関数locateはその名の通りカーソルを移動させますが、リスト中ではＡＤＭ－

３ＡのオーダにしてありますのでこのままではＦＭ－７、８では動きません。次のように変更して下さい。

```
locate( x, y )
    int  x, y;
{
    static char locbuf[3] = { 0x12, 0, 0 };

    locbuf[1] = x;
    locbuf[2] = y;
    fwrite( locbuf, sizeof(locbuf), 1, stdout );
}
```

このchgbuf.cのファイルはそのままで他のプログラムに流用できます。なぜなら、配列bufをスクリーンエディットで修正するというだけのものにすぎないのですから。その場合は座標を合わせておく必要がありますが。

```
/*    file screen editor    */
/*    usage: fedit file <sctor>.    */

#include <stdio.h>

#define  ERR (-1)
#define  YES 1
#define  NO  0

FILE  *fp;
long  maxsct;
char  *fname;


main( argc, argv )
     int  argc;
     char *argv[];
{
     FILE *fopen();
     long atol(), flength();
     long sct = 0;

     if ( argc != 2 && argc != 3 ) {
          fprintf( stderr , "usage: fedit filename <sector>.¥n" );
          return;
     }
     if ( argc > 2 )
          sct = atol( argv[2] );
     if ( ( fp = fopen( fname = argv[1] , "r+" ) ) == NULL ) {
          fprintf( stderr , "can't open.¥n" );
          return;
     }
     if ( sct > ( maxsct=(flength(fp)+BUFSIZ)/BUFSIZ-1 ) ) {
          fprintf( stderr, "sector number too big.¥n" );
          return;
     }
     setbuf( stdin, NULL );
     setbuf( stdout, NULL );
     allowecho( NO );
     edit( sct );
     allowecho( YES );
     fclose( fp );
}
```

```
long flength( fp )
     FILE *fp;
{
     long ftell();

     fseek( fp, 0L, 2 );
     return( ftell( fp ) );
}
#include <sgstat.h>

allowecho( echo )
     int  echo;
{
     static char buffer[32];
     static int  echomode = YES;

     if ( echo ) {
          getstat( 0, 1, buffer );
          buffer[4] = echomode;
          setstat( 0, 1, buffer );
     } else {
          getstat( 0, 1, buffer );
          echomode = buffer[4];
          buffer[4] = NO;
          setstat( 0, 1, buffer );
     }
}
```

```
/*    file edit    */

#include <stdio.h>

#define YES 1
#define NO  0
#define ERR (-1)

#define BACK 0x19
#define NEXT 0x1a

char buf[BUFSIZ];


edit( sct )
     long sct;
{
     extern long maxsct;
     int  exkey;

     pfinit();
     while ( readbuf( sct ) != ERR ) {
          if ( ( exkey = chgbuf() ) == EOF )
               break;
          writebuf( sct );
          if ( exkey == BACK && sct > 0 ) {
               sct--;
          } else if ( exkey != BACK  && sct < maxsct ) {
               sct++;
          } else
               continue;
     }
     locate( 0, 22 );
     fprintf( stderr, "fed abort.¥n" );
}


int readbuf( sct )
     long sct;
{
     extern char buf[];
     extern FILE *fp;
     extern long maxsct;
     int  i;
```

```
        if ( sct > maxsct )
             return( ERR );
        fseek( fp, sct*BUFSIZ, 0 );
        for ( i = 0; i < BUFSIZ; i++ )
             if ( ( buf[i] = getc( fp ) ) == EOF )
                  buf[i] = 0;
        dump( buf, sct );
        return( NO );
}

#define CLS 0x0c
#define LINSIZ 16
#define toprint( c ) ( ( (c) < 0x20 || (c) == 0x7f || (c) == 0xff )? '.': (c) )

dump( p, sct )
        char *p;
        long sct;
{
        extern char *fname;
        int  pt, ln;

        putchar( CLS );
        printf( "file editor¥n¥nfile = %s  sector = %ld¥n", fname, sct );
        printf( "       0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f" );
        printf( "     0 2 4 6 8 a c e¥n" );
        printf( "       ----- ----- ----- ----- ----- ----- ----- -----" );
        printf( "     ----------------¥n" );

        for ( ln = 0; ln < BUFSIZ; ln += LINSIZ ) {
             printf( "%02x : ", ln );
             for ( pt = 0; pt < LINSIZ; pt++ )
                  printf ( "%02x ", p[ln+pt] & 0xff );
             printf( " : " );
             for ( pt = 0; pt < LINSIZ; pt++ )
                  putchar( toprint( p[ln+pt] ) );
             putchar( '¥n' );
        }
}


writebuf( sct )
        long sct;
{
        extern char buf[];
        extern FILE *fp;
        int  i;

        if ( writeok() ) {
             fseek( fp, sct*BUFSIZ, 0 );
             for ( i = 0; i < BUFSIZ; i++ )
                  if ( putc( buf[i], fp ) == EOF )
                       break;
        }
}


int  writeok()
{
        int  c;

        for ( ;; ) {
             locate( 0, 23 );
             printf( "Do you want to write this sector ?(y/n) " );
             if ( putchar( c = getchar() ) == 'y' || c == 'Y' ) {
                  printf( "    writing..¥n" );
                  return( YES );
             } else if ( c == 'n' || c == 'N' ) {
                  printf( "    Not write..¥n" );
                  return( NO );
             }
        }
}
```

```
/*    change buffer     */

#include <stdio.h>

extern buf[];
extern FILE  *fp;
extern long  maxsct;
extern char  *fname;

#define YES 1
#define NO  0
#define ERR (-1)

#define RIGHT  0x1c      /* right */
#define LEFT   0x1d      /* left  */
#define UP     0X1e      /* up    */
#define DOWN   0x1f      /* down  */
#define BACK   0x19      /* backword sector ^Y */
#define NEXT   0x1a      /* next sector     ^Z */
#define MOVASC 0x06      /* to ascii mode   ^F */
#define MOVHEX 0x02      /* to hex mode     ^B */
#define BS     0x08      /* back space */
#define SKIP   ' '       /* skip */
#define CR     0x0d      /* line feed */
#define HOME   0x0b      /* home */


static int pt;          /*    buffer pointer          */

int chgbuf()
{
     int  c;

     pt = 0;
     for ( ;; ) {
          if ( ( c = hexchange() ) != MOVASC )
               break;
          if ( ( c = ascchange() ) != MOVHEX )
               break;
     }
     return( c );
}



int hexchange()
{
     int  high = YES;
     int  c;

     hexlocate( pt, high );
          switch ( c ) {
          case RIGHT:
          case SKIP:
               if ( high )
                    hexlocate( pt, high=NO );
               else if ( pt+1 < BUFSIZ )
                    hexlocate( ++pt, high=YES );
               break;
          case LEFT:
          case BS:
               if ( high && pt > 0 )
                    hexlocate( --pt, high=NO );
               else if ( !high )
                    hexlocate( pt, high=YES );
               break;
          case UP:
               if ( pt-16 >= 0 )
                    hexlocate( pt-=16, high );
               break;
          case DOWN:
               if ( pt+16 < BUFSIZ )
                    hexlocate( pt+=16, high );
               break;
          case HOME:
            hexlocate( pt=0, high=YES );
            break;
        case CR:
            if ( (pt&=~0x0f)+16 < BUFSIZ )
                pt += 16;
            hexlocate( pt, high=YES );
            break;
        case MOVASC:
        case BACK:
        case NEXT:
            return( c );
        default:
            if ( ( c = ahtoi( c ) ) == ERR )
                break;
            if ( high ) {
                buf[pt] = c * 16 + ( buf[pt]&0x0f );
                putbuf( pt );
                hexlocate( pt, high=NO );
            } else {
                buf[pt] = c + ( buf[pt]&0xf0 );
                putbuf( pt );
                if ( pt+1 < BUFSIZ )
                    hexlocate( ++pt, high=YES );
                else
                    hexlocate( pt, high );
            }
            break;
        }
    return( EOF );
}


hexlocate( pt, high )
    int  pt, high;
{
    locate( 5+(pt%16)*3+(high?0:1), 5+(pt/16) );
}



int ahtoi( c )
    int  c;
{
    if ( c >= '0' && c <= '9' )
        return( c - '0' );
    else if ( ( c = tolower( c ) ) >= 'a' && c <= 'f' )
        return( c - 'a' + 10 );
    else
        return( ERR );
}
int ascchange()
{
    int  c;

    while ( ( c = getchar() ) != EOF )
        switch ( c ) {
        case RIGHT:
            if ( pt+1 < BUFSIZ )
                asclocate( ++pt );
            break;
        case LEFT:
        case BS:
            if ( pt > 0 )
                asclocate( --pt );
            break;
        case UP:
            if ( pt-16 > 0 )
                asclocate( pt-=16 );
            break;
        case DOWN:
            if ( pt+16 < BUFSIZ )
                asclocate( pt+=16 );
            break;
        case HOME:
            asclocate( pt=0 );
            break;
```

```
                case CR:
                        if ( (pt&=~0x0f)+16 < BUFSIZ )
                                pt += 16;
                        asclocate( pt );
                        break;
                case MOVHEX:
                case BACK:
                case NEXT:
                        return( c );
                default:
                        if ( c >= 0x20 && c != 0x7f && c != 0xff ) {
                                buf[pt] = c;
                                putbuf( pt );
                                asclocate( ( pt+1 < BUFSIZ )? ++pt: pt );
                        }
                        break;
                }
        return( EOF );
}



asclocate( pt )
        int  pt;
{
        locate( 56+(pt%16), 5+(pt/16) );
}

#define toprint( c ) ( ( (c) < 0x20 || (c) == 0x7f || (c) == 0xff )? '.': (c) )

putbuf( pt )
        int  pt;
{
        int c;

        c = buf[pt] & 0xff;
        hexlocate( pt, YES );
        putchar( itoh1( c / 16 ) );
        putchar( itoh1( c % 16 ) );
        asclocate( pt );
        putchar( toprint( c ) );
}



locate( x, y )
        int  x, y;
{
        static char locbuf[4] = { 0x1b, '=', 0, 0 };

        locbuf[2] = y + 0x20;
        locbuf[3] = x + 0x20;
        fwrite( locbuf, sizeof(locbuf), 1, stdout );
}



int itoh1( c )
        int  c;
{
        if ( c >= 0 && c <= 9 )
                return( c + '0' );
        else if ( c >= 10 && c <= 15 )
                return( c - 10 + 'a' );
        else
                return( '?' );
}
```

以上の3つのファイルをコンパイルするには次のようにします。

まず、ソースファイルを、

```
fed.c　　edit.c　　chgbuf.c
```

のように名前を付けます。

```
OS9:cc2　fed.c　edit.c　chgbuf.c
```

この時、mainを含むファイルを最初に置かなければなりません。

あるいは、chgbuf.cをあとで使用する可能性があるのなら、

```
OS9:cc2　chgbuf.c　－r
OS9:cc2　fed.c　edit.c　chgbuf.r
```

として、chgbuf.rを保存しておきます。そうすれば使うたびにCのソースからコンパイルする手間がはぶけます。

最後に、デバッグ用に使えるcoredump.cを挙げておきます。第一引数は256バイトのバッファへのポインタ、第二引数はアドレスとして表示するためのlong integerの値です。longなのは一般にファイルポインタは4バイトで表されるためです。したがって、coredumpを使用する関数内では次のように宣言しておかなければなりません。

```
long　*coredump()；
```

```
#include <stdio.h>
#include <ctype.h>
#define CMASK 0xff

coredump( p, pos )
    char p[16][16];
    long pos;
{
    int  i, j;

    pflinit();
     printf( " addr    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f " );
     printf( "  0 2 4 6 8 a c e¥n" );
     printf( "------  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- " );
     printf( "  ----------------¥n" );
     for ( i = 0; i < 16; i++ ) {
         printf( "%06lx  ", pos );
         pos += 16;
         for ( j = 0; j < 16; j++ )
```

```
            printf( "%02x ", p[i][j] & 0x00ff );
        printf( "  " );
        for ( j = 0; j < 16; j++ )
            putchar( isprint( p[i][j] )? p[i][j]: '.' );
        putchar( '¥n' );
    }
}
```

## 2—2—11　cp

cpは単なるコピープログラムです。ＯＳ－９の付属のものに若干の機能を付け加えてありますが、逆に機能縮小した点もあります。

まずオプション“－u”と“－w＝’を付けました。‘－u’はバッファリングをいっさいしないというモードで、１バイトずつコピーしてゆきます。ターミナル等を指定した場合に効果があります。‘－w＝’はファイルの転送先のディレクトリを指定するもので、複数のファイルを一度にコピーする場合に使います。例えば、

```
OS9 : cp   file1   file2   file3   －w＝/d1/tempdir
```

とすると/d1/tempdirというディレクトリ上に、‘file1’, ‘file2’, ‘file3’というファイルをコピーします。この時‘－w＝’で指定するものはディレクトリでなければなりません。

また‘－w＝’が無くてファイルを２つ指定した時は通常の意味でのコピーを、１つ指定した時はそのファイルから標準出力へコピーし、１つも置かなかった時は、標準入力から標準出力へコピーします。まとめますと次のような書き方が許されることになります。

```
OS9 : cp   file1   file2   file3   －w＝/d1/tempdir
OS9 : cp   file1   file2
OS9 : cp   file1
OS9 : cp
```

‘−w=’の指定をせずにファイルを3つ以上置くのは間違いですし、ファイルが1つまた2つでも、‘−w=’の指定があるのとないのでは意味が異なります。

いつものように、‘−w=’の指定はファイル名の前にあっても後にあってもよいことにしましょう。そのためにはまたオプションの解析をする関数を別にしておきます。オプションで指定するディレクトリ名を表すには1個のポインタで事足りますので、ポインタ wild に設定することにします。また指定が無かった場合はNULLにしておきます。

なお、書き込みファイルのクリエイトには

```
fopen(  p,  “w”  )
```

のように、“w”でオープンしていますので、もし同じ名前のファイルが存在した場合、いったんそのファイルを削除してしまいますので特に注意して下さい。

関数 option がオプションの設定をします。またファイル名の個数によってすることが変わってくることから、option の中でファイル名の数を数え、その値を返すことにします。もしオプションに間違いがあった場合はファイルの数の替わりにERRを返します。

main は wild があるかないかとファイルの数によって動作が違いますので、if-else で場合分けします。wild がある場合はファイル名の数は関係ありませんので wild を先に調べる必要があり、それから、ファイルの数によって分けます。

また下請けの関数として、filecopy というものを用意しておきます。これはフィルタポインタ fp1 で表されるファイルから、fp2 で表されるファイルへコピーするものです。実質的なコピーはこの関数が行い、他はそのための環境を整えるだけのものと思って差しつかえありません。したがってコピーのスピードはこの関数の出来によって決定的に違ってきますので、かなりの手間をかけるだけの価値はあります。が、ここでは最も簡単なものを挙げておきます。

もし‘−u’オプションがあった場合、setbuf 関数でバッファリングを禁止しておく必要があります。バッファリングの禁止はファイルのオープンの直後に行う必要がありますので filecopy の中に入れてあります。

さて copy1, copy2, copyw という似たような関数が並んでいますが、それぞれ

ファイルから標準出力、ファイルからファイル、そして‘−w=’の指定がある場合となっています。このように分けてあるのはそれぞれでファイル名の探し方が少し違うからです。ようするに、第一のファイル名を探すにはargv[1]では駄目なのです。どこにオプションがあるか分からないのですから。

copywではコピー先のファイル名を作ってやらなければなりません。それには‘−w=’で指定したディレクトリ名の後ろにコピー元のファイル名を接げてやるだけでは十分ではありません。コピー元のファイルには一般にパスリストで与えられますが、コピー先のファイル名として接げてやらなければならないのは、その一番下のレベルの部分だけなのです。つまりパスリストの一番最後の‘／’の位置を知る必要があるのです。

標準関数にはそのためにぴったりのものがあります。rindexがそれで、文字列の後ろからキャラクタを探し、みつかった位置をポインタで返します。また、みつからなかった場合はＮＵＬＬを返します。wildが指す文字列と、rindexが返す文字列（プラス1）を、間に“／”を狭んで接げればよいわけです。

```
/*    cp  -- copy file(s), non-buffering version    */

/*    usage: cp <opt> {file} <opt>.
        opt: -w=DIR  destination directory.
             -u      un-buffuring.         */

#include <stdio.h>

#define YES 1
#define NO  0
#define ERR (-1)

char *wild = NULL;
int  unbuf = NO;


main(argc, argv)
    int  argc;
    char **argv;
{
    char *p;
    int  fcount;

    if ( ( fcount = option( argc ,argv ) ) == ERR ) {
        usage();
        return;
    }
    if ( wild )
        while ( --argc ) {
            if ( *(p=*++argv) == '-' )
                continue;
            copyw( p );
        }
    else if ( fcount == 2 )
        copy2( argc, argv );
    else if ( fcount == 1 )
        copy1( argc, argv );
    else if ( fcount == 0 )
```

```
        filecopy( stdin, stdout );
    else
        usage();
}


int option( argc, argv )
    int  argc;
    char **argv;
{
    char *p;
    int  fcount = 0;
    while ( --argc )
        if ( *(p=*++argv) == '-' ) {
            switch ( p[1] ) {
            case 'w':
            case 'W':
                if ( p[2] == '=' )
                    wild = p + 3;
                else
                    return( ERR );
                break;
            case 'u':
            case 'U':
                unbuf = YES;
                break;
            default:
                return( ERR );
            }
        } else
            fcount++;
    return( fcount );
}


usage()
{
    fprintf( stderr, "usage: cp <opt> {file} <opt>.¥n" );
    fprintf( stderr, "  opt: -w=DIR  destination.¥n" );
    fprintf( stderr, "       -u      un-buffering.¥n" );
}


filecopy( fp1, fp2 )
    FILE *fp1, *fp2;
{
    int c;

    if ( unbuf ) {
        setbuf( fp1, NULL );
        setbuf( fp2, NULL );
    }
    while ( ( c = getc( fp1 ) ) != EOF )
         putc( c , fp2 );
}


copy2( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp1, *fp2, *fopen();
    char *p;

    while ( --argc ) {
        if ( *(p=*++argv) == '-' )
            continue;
        if ( ( fp1 = fopen( p, "r" ) ) == NULL ) {
            fprintf( stderr, "can't open ¥"%s¥".¥n", p );
            return;
        } else
            break;
    }
    while ( --argc ) {
        if ( *(p=*++argv) == '-' )
            continue;
        if ( ( fp2 = fopen( p, "w" ) ) == NULL ) {
            fprintf( stderr, "can't create ¥"%s¥".¥n", p );
            fclose( fp1 );
            return;
        } else
            break;
    }
    filecopy( fp1 , fp2 );
    fclose( fp1 );
    fclose( fp2 );
}

copy1( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();
    char *p;

    while ( --argc ) {
        if ( *(p=*++argv) == '-' )
            continue;
        if ( ( fp = fopen( p, "r" ) ) == NULL ) {
            fprintf( stderr, "can't open ¥"%s¥".¥n", p );
            return;
        } else
            break;
    }
    filecopy( fp, stdout );
    fclose( fp );
}



copyw( p )
    char *p;
{
    FILE *fp1, *fp2, *fopen();
    char dest[BUFSIZ];

    fnamec( p, dest );
    if ( ( fp1 = fopen( p, "r" ) ) == NULL ) {
        fprintf( stderr, "can't open ¥"%s¥".¥n", p );
        return;
    } else if ( ( fp2 = fopen( dest, "w" ) ) == NULL ) {
        fprintf( stderr, "can't create ¥"%s¥".¥n", dest );
        fclose( fp1 );
        return;
    }
    filecopy( fp1 , fp2 );
    fclose( fp2 );
    fclose( fp1 );
}



fnamec( p, dest )
    char *p, *dest;
{
    char *s, *rindex();

    strcpy( dest, wild );
    strcat( dest, "/" );
    if ( s = rindex( p, '/' ) )
        strcat( dest, ++s );
    else
        strcat( dest, ■ );
}
```

前にも述べたように filecopy を改善することは大きな意味があります。ここではその一つの方法としてバッファリングについて述べます。

バッファリングとは大きなバッファをメモリ上に用意しておいて、一度に大量に読みこみ、大量に書きこむというものです。問題はバッファをどこに確保するかですが、ＯＳ－９ではプロセスの起動時にデータエリアが与えられることになっており自由に使えるうえ、そのサイズをコマンド起動時に指定できるようになっています。これを使わない手はありません。

データエリアは自由に使える部分の他に、スタック等の領域も含まれます。そこで、自由に使える部分のアドレスおよびそのサイズを知る必要があります。それには関数 freemem と ibrk を使用します。freemem は未使用のデータエリアのサイズを返し、ibrk はサイズを指定してそのエリアの使用権を確保するとともにベースアドレスを返します。

肝心のファイルアクセスの方法ですが、getc を使った場合および fread と fwrite を使った場合を比較してみたところ、思った程の効果が無かったどころか、何もしなかった場合よりもスピードが落ちてしまいました。これは getc のオーバーヘッドによるものと思われます。

そこで read と write を使う方法に切り換えました。この２つはファイルポインタではなくパス番号を使う関数ですが、ファイルポインタからパス番号を得るのは簡単です。fileno 関数（実はマクロ）を使えばよいのです。

read は指定したバイト数を読み込み実際に読み込んだバイト数を返します。また write は書き込むべきバイト数を指定するのですから read が返した値でそのまま write してしまえばよいのです。read はＥＯＦあるいはエラーの際にはゼロを返しますから、それをループの終了条件にします。

以上を組み込むためには filecopy の書き換えの他に、main の中でバッファを求めることが必要です。

main、バッファを求める getfree および filecopy を挙げます。

```
/*    cp  -- copy file(s), self-buffering version    */

/*    usage: cp <opt> (file) <opt>.
        opt: -w=DIR  destination.
             -u      un-buffering.        */

#include <stdio.h>
```

```
#define YES 1
#define NO  0
#define ERR (-1)

char *wild = NULL;
int  unbuf = NO;
char *base;
int  buffersize;


main( argc, argv )
    int  argc;
    char **argv;
{
    char *p;
    int  fcount;

    if ( ( fcount = option( argc ,argv ) ) == ERR ) {
        usage();
        return;
    }
    getfree();
    if ( wild )
        while ( --argc ) {
            if ( *(p=*++argv) == '-' )
                continue;
            copyw( p );
        }
    else if ( fcount == 2 )
        copy2( argc, argv );
    else if ( fcount == 1 )
        copy1( argc, argv );
    else if ( fcount == 0 )
        filecopy( stdin, stdout );
    else
        usage();
}



getfree()
{
    if ( ( buffersize = freemem() & (~0xff) ) == 0 )
        base = NULL;
    else if ( ( base = ibrk( buffersize ) ) == ERR || base == NULL )
        base = NULL;
}



filecopy( fp1, fp2 )
    register FILE *fp1;
    FILE *fp2;
{
    int  size, pn1, pn2, c;

    if ( unbuf ) {
        setbuf( fp1, NULL );
        setbuf( fp2, NULL );
        while ( ( c = getc( fp1 ) ) != EOF )
            putc( c, fp2 );
    } else {
        pn1 = fileno(fp1);
        pn2 = fileno(fp2);
        while ( size = read( pn1, base, buffersize ) )
            if ( write( pn2, base, size ) == ERR )
                break;
    }
}
```

どのくらい速くなったか、またcopyと比較してみてどうかを調べてみました。ファイルは適当に大きいものということでos9bootを使用し、次のようにして所

要時間を計りました。

OS9 : date t; cp /nd0/os9boot /nd1/os9boot; date t

もちろん date とテストプログラムはロードしておきます。

| | |
|---|---|
| non-buffering version の cp | |
| OS9 : cp /nd0/os9boot /nd1/os9boot | 1：08 |
| OS9 : cp /nd0/os9boot /nd1/os9boot #50k | 1：08 |
| buffering version の cp | |
| OS9 : cp /nd0/os9boot /nd1/os9boot | 0：50 |
| OS9 : cp /nd0/os9boot /nd1/os9boot #50k | 0：28 |
| copy | |
| OS9 : copy /nd0/os9boot /nd1/os9boot | 0：16 |
| OS9 : copy /nd0/os9boot /nd1/os9boot #50k | 0：15 |

なお、'−u'オプションを指定した場合3：25かかったことを付け加えておきます。

もし filecopy をアセンブラで書いたら、もう少し速くなるのではないかと考えたのが次のリストですが、結果はまったく変わりませんでした。参考までに挙げておきます。

```
filecopy( fp1, fp2 )
    register FILE *fp1;
    FILE *fp2;
{
    int  size, c;

    if ( unbuf ) {
        setbuf( fp1, NULL );
        setbuf( fp2, NULL );
        while ( ( c = getc( fp1 ) ) != EOF )
            putc( c, fp2 );
    } else
        asmcopy( fileno( fp1 ), fileno( fp2 ) );
}
```

```
#asm

 ifp1
 use /d0/defs/os9defs.a
 endc

asmcopy:
 pshs x,y
 ldx base,y
 ldy buffersi,y buffersize
 sty ,s

fcploop
     lda 7,s pn1
     ldy ,s buffersize
     os9 I$READ
     bcs fcperr
     lda 9,s pn2
     os9 I$WRITE
     bcs fcperr
     bra fcploop

fcperr
     cmpb #E$EOF
     bne fcpexit
     clrb
fcpexit
     clra
     puls x,y,pc

 #endasm
```

## 2―2―12　calcu

ＯＳ－９のShell上から呼び出せる卓上計算機を作成しました。これはコマンドとして使えるほか、計算モードに入って連続的に計算することが可能です。プログラムはPascal（Microware社製）で表現されていますが、Basic09に移植することはそれほど苦労を要しないでしょう。プログラムはコンパイラのパーザ(構文解析プログラム）に似ていますが、これは、数式を解釈していくという卓上計算機の機能というものが、コンパイラの機能の一部ですから、当然の結果とも言えるでしょう。

―使用法―

calcuは次に示すような２通りの使用方法ができます。

・コマンドとして使う（１度に１回限り）

・計算モードに入って使う（連続的に使える）

コマンドとして使うときは、コマンドの後ろにコロン‘：’で区切って数式を書きます。

```
calcu  :  数式
```

例えば、

```
calcu  :  34 * 21.0+(23/$22)
```

計算モードにはいるときは、コマンドの後ろにコロンで区切った数式を書かないようにします。計算モードに入ると‘calcu:’というプロントが出てきますので、その後ろに数式を入力するようにします。

```
calcu
```

入力と出力は、標準入力と標準出力に対して行っていますので、リダイレクトが可能です。

```
calcu  </d0/file  >/p  #40k
```

この簡易計算機では、数式を計算するほかに、変数や関数を扱うことができます。変数に値を記憶するときは、代入文を使用します。また、関数は現在のところ組込み関数として３２個備えています。また、内部的には数値は全て実数として扱われており、結果は実数として出力されます（必ず小数点‘.’がつきます)。それでは、定数・演算子・変数・関数の順で説明していきます。

＊定数

この簡易計算機で扱う数は、整数と実数の区別はなく、すべて実数として扱われます。そしてその範囲は約２．９＊１０＾(－３９)から１．７＊１０＾３８までです。精度は約9$\frac{1}{2}$桁の有効桁数を持っています。また、入力する際には‘E’(小文字でも可）をつけた指数表現も可能です。なお、小数点‘．’だけの場合は0.0として扱われます。数の間に空白‘　’がいくらあっても構いません。例えば、

```
８２７０８７９５
１２３４．５６７
０．８７３４５６
－７２３４ｅ－２２
７３４．Ｅ１２
１  ２  ３  ４      (＝１２３４として扱います)
５６２３．

                    (＝０．０として扱います)
```

また、‘$’を前置すれば、１６進数として入力することができます。ただしこのときは、小数部分を１６進数で入れることはできません。

```
＄７４ａｆ
＄０４ＡＤＦＢ
```

＊演算子

　ここで述べる演算子とは、２項演算子のことです。演算子は下表の優先順序と評価順序で計算されていきます。

| 優先順位 | 演算子 | 評価順序 |
|---|---|---|
| 1 | ^ | 左──→右 |
| 2 | ＊　／　￥　％ | 左──→右 |
| 3 | ＋　－ | 左──→右 |

個々の演算子について、説明します。

^　：べき乗を示します。　例　４^２　→答１６．０
＊　：乗算を示します。　例　３．２＊３　→答９．６
／　：除算を示します。　例　８／１６　→答０．５
￥　：整数除算を示します。　例　８￥３　→答２．０
％　：剰余を示します。　例　８％５　→答３．０
＋　：加算を示します。　例　８＋４　→答１２．０
－　：減算を示します。　例　１９８４－１０１４　→答９７０．０

なお、￥、％は実数に対しても扱えますし、負の数にも対応します。

８．９￥２．６　→答　３．０
８．９％０．７　→答　０．５
－４　￥１．３　→答－２．０
－１３％（－４）　→答　１．０
２２　％（－４．５）→答－４．０

　符号として＋と－は２項演算子として解釈されますので、優先順位も一番低いレベルとなります。したがって、次のようなものは括弧をつけなければなりません。つけていない場合はエラーになります。

２５^－２　（誤り）
２５^（－２）　（正解）

### ＊変数

計算モードのときは、変数が使えます。コマンドとして使うときも、変数は使用可能ですが、値が残らないので意味がありません。変数として使えるのは、‘A’～‘Z’の２６個です。大文字と小文字の区別はありません。変数に代入するときは次のような書式になります。

> 変数名　＝　数式
>
> 　　（変数名は‘A’～‘Z’のうちどれか）

例えば、

```
a＝７２８３＋８３２４
X＝６６．３４／８７４．２
X＝７４５＋３５４／s＊D
X＝X＋２
```

変数の値を知りたいときは、知りたい変数名だけを入力すれば見ることができます。また、“？”を入力すれば、すべての変数の値を見ることができます。

### ＊関数

関数は、３２個あります。関数を使うときは次の書式で書きます。

関数名（引数）

関数名は、次のうちのどれかです。

| 関数名 | 関数の内容 |
|---|---|
| sin | サイン |
| cos | コサイン |
| tan | タンジェント |
| sinh | ハイパーボリックサイン |

| | |
|---|---|
| cosh | ハイパーボリックコサイン |
| tanh | ハイパーボリックタンジェント |
| arcsin | アークサイン |
| arccos | アークコサイン |
| arctan | アークタンジェント |
| arcsinh | ハイパーボリックアークサイン |
| arccosh | ハイパーボリックアークコサイン |
| arctanh | ハイパーボリックアークタンジェント |
| sec | セカント |
| csc | コセカント |
| cot | コタンジェント |
| sech | ハイパーボリックセカント |
| csch | ハイパーボリックコセカント |
| coth | ハイパーボリックコタンジェント |
| arcsec | アークセカント |
| arccsc | アークコセカント |
| arccot | アークコタンジェント |
| arcsech | ハイパーボリックアークセカント |
| arccsch | ハイパーボリックアークコセカント |
| arccoth | ハイパーボリックアークコタンジェント |
| exp | ｅ（自然対数の底）を底とした指数関数 |
| log | 常用対数（底は１０．０） |
| ln | 自然対数（底はｅ＝２．７１８２８・・・・） |
| sqr | 二乗 |
| sqrt | 平方根 |
| rad | ラジアン（引数をラジアン単位に直す） |
| deg | デグリー（引数を度単位に直す） |
| int | 引数の整数部分を返す |

関数に対する引数の値がおかしい場合は、エラーになります。

**＊その他**

答の表示は、変数へ代入したときも行います。表示する内容は次のようになります。

```
answer＝答　　hex＝答（１６進法）
```

ただし、１６進法の表示は整数の部分に対してのみ行われます。また、１０進法の答の表示は、その値の絶対値が１．０ｅ＋１０～１．０ｅ－１０の範囲にあるものは、見やすい形に直されます。このときの表示桁数は、１０桁です。

計算には誤差が含まれます。Microware の Pascal 自体は $9\frac{1}{2}$ 桁ぐらいの精度ですが、この計算機プログラムでは何回も計算を繰りかえしていますのでそれ以下となり、だいたい７～８桁の精度しか見込めないようです。

エラーが起こった場合は、答は表示されません。かわりに、エラー内容と起こった場所が次のように表示されます。

```
error　エラー番号　occured at　カラム番号　columns
```

カラム番号というのは、何文字目かということを示しています。エラー番号は次のように決めています。

| エラー番号 | エラー内容 |
|---|---|
| 1 | divided by zero |
| 2 | syntax error |
| 3 | overflow |
| 4 | ')' expected |
| 5 | function range error |
| 6 | undefined function |

また、括弧のネスティングが多すぎた場合はPascalのStack Overflowが出ます。

—プログラムの解説—

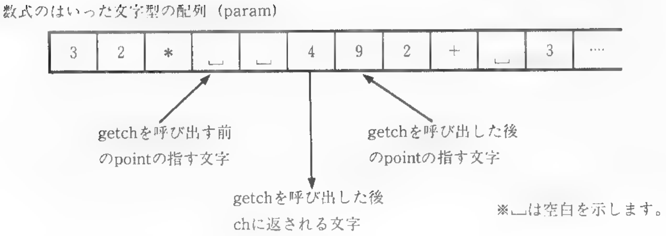
数式を評価していくのには、数式を一度逆ポーランド記法に変換してから評価していく方法もありますが、ここでは優先順位を考慮しながら逐次評価していくという方法をとりました。入力された数式を逐次評価していくのは、構文解析プログラム(パーザ)、の機能と同じようなものです。そこで、この計算機のプログラムはパーザに似せて構成しました。それでは、最終的なプログラムを説明するために、簡単なモデルから出発して、それに機能を付加していくという形で話を進めていきましょう。

＊数式の読込み

まず、整数の四則演算（優先順位を考えない）だけができる一番簡単なモデルを考えてみます。入力された数式（文字列）はparamという名前の配列（長さはLENGで、空白で初期化されている）に入っているとします。ここで必要なのは、１文字読込んでくるルーチンです。また、読み込んできた文字によって動作が異なりますから、１文字分先読みするということも必要になってきます。そこで、文字列に対するポインタを２つ用意して、一方を先読み用に使います。

では、getchという名前の一文字読み込むルーチンを設計してみます。このルーチンは、文字列の中からポインタ(point)が指している場所の文字をchに読み込み、ポインタを次の場所に更新させます（読み込んだ文字が空白ならスキップします)。

```
procedure getch ( var ch:char; var point:integer );
begin
     if point<=LENG then
          repeat
               ch   :=param[point];
               point:=point+1
          until (ch<>' ') or (point>LENG)
     else
          ch:=' '
end;
```



※␣は空白を示します。

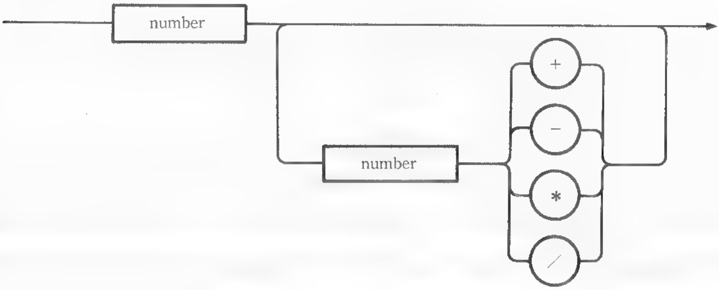
getchを呼び出したときの例

このルーチンと2つのポインタ（order、pointerでpointerは先読み用）を使って最初のモデルを実現してみましょう。実現する上でその他に、数式を入力してくれるプロシジャinputparamと、整数を解釈するファンクションnumberを仮定します。このファンクションには、ポインタを渡します。戻ってきたときのポインタは、整数文字列の次の場所を指しています。このモデルでは、不正入力のあったときや、0で割ったときのエラー処理は考えないことにします。（このモデルでは、先読み用のポインタpointerはあまり活用されていません。）

```
program prototype1 (input,output);

     const LENG=250;

     var  ans   : real;
          param: array [0..LENG] of char;
          order,pointer: integer;
          ch    : char;

     {Here are definitions of inputparam and number and getch}
     begin
          inputparam;
          order:=0;  pointer:=0;
          getch(ch,pointer);
          if ch<>' ' then
          begin
               ans:=number(order);
               getch(ch,order);
               while ch in ['+','-','*','/'] do
               begin
                    case ch of
                         '+' : ans:=ans+number(order);
                         '-' : ans:=ans-number(order);
                         '*' : ans:=ans*number(order);
                         '/' : ans:=ans/number(order)
                    end;
                    getch(ch,order)
               end;
               write ('answer=',ans);
          end
```

```
        end.
```

このプログラムが数式にどんな要素を期待しているかを明確にするには、数式の構文を構文グラフにしてみるとわかりやすいでしょう。このプログラムは、次の構文グラフが示すように、最初に数 (number) があり、その後に演算子＋数の繰返しがあることを期待しているのです。



prototype 1の構文グラフ

＊演算の優先順位

最初に作ったプログラムでは、演算子の優先順位が考慮されていなかったため、例えば8＋2＊40という数式に対する答が400となってしまいます。ここで優先順位を実現するために、トップダウン解析という手法を用いることにします。これはプログラムのトップダウン設計と同じで、数式の構文を段階的に詳細化していくものです。

一番上位のレベルでは、加算と減算だけについて扱います。その次のレベルでは、乗算と除算について扱います。一番下のレベルで整定数を評価します。それぞれのレベルは、サブルーチンとして構成することができます。最上位のレベルのサブルーチンを expression、その次を term、最下位のサブルーチンを number と名付けます。前述の prototype1 の中で number というファンクションを仮定しました。概念を説明はしませんでしたが、すでにトップダウンの手法を取り入れていたのです。

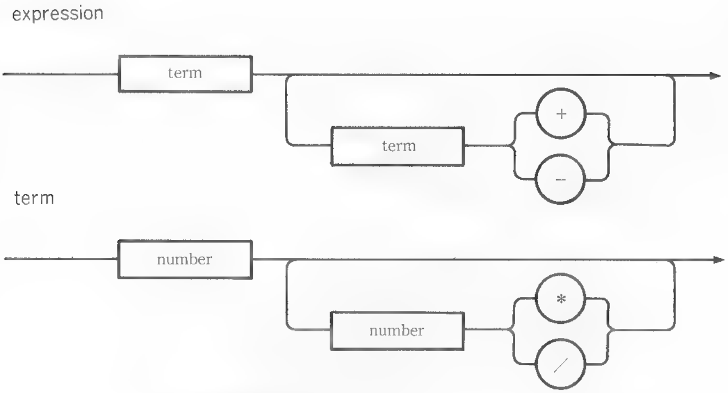
それでは、次のモデル（プログラム）を設計してみます。今まで述べてきた各

レベルのサブルーチンは、数式の評価結果を値として返してくるファンクションとして作成していきます。

```
program prototype2 (input,output);

     {same declaration as prototype1}

function term ( var order:integer ):real;
     var  pointer: integer;
          ans    : real;
          ch     : char;
     begin
          ans    :=number(order);
          pointer:=order;     getch(ch,pointer);
          while ch in ['*','/'] do
          begin
               order:=pointer;
               case ch of
                    '*' : ans:=ans*number(order);
                    '/' : ans:=ans/number(order)
               end;
               pointer:=order;      getch(ch,pointer)
          end;
          term:=ans
     end;

function expression ( var order:integer ):real;
     var  pointer: integer;
          ans    : real;
          ch     : char;
     begin
          ans    :=term(order);
          pointer:=order;     getch(ch,pointer);
          while ch in ['+','-'] do
          begin
               order:=pointer;
               case ch of
                    '+' : ans:=ans+term(order);
                    '-' : ans:=ans-term(order)
               end;
               pointer:=order;     getch(ch,pointer)
          end;
          expression:=ans
     end;

begin
     inputparam;
     order:=0;
     write('answer=',expression(order))
end.
```

このモデルに対応する構文グラフは次のように2段階（numberも入れれば3段階）に分かれます。このように、後から優先順位の高いものがきても、下のレベルで先にそれを計算してしまいますから、優先順位を実現することができます。（次頁の図参照）

この2番目のモデルで、実際のプログラムの主要な概念を説明したことになります。



prototype 2の構文グラフ

＊実際のプログラム

実際のプログラムでは、今まで述べてきたような構文解析の処理の他に、エラーの処理や変数の処理、入出力の処理を行うことが必要です。また、複雑な構文グラフに対して、評価していかなければなりません。

数式は、多数のレベルに分けて評価されていきます。それぞれのファンクションと評価レベルとの対応を説明しましょう。各ファンクション名は、構文グラフ上に掲げた名前とほぼ一致させています。

statement　：　入力された文字列が、変数への代入文か、ただの数式かを判断し、代入文の場合は数式を評価した後で変数に代入しています。数式の評価は、expressionに任せています。評価がうまくいけば答を、評価している最中で何らかのエラーが起きた場合はエラーの表示を行います。

expression　：　加減算を評価します。expressionは、左括弧が現れたときに呼び出されますので、右括弧がくるか、もう文字列がない状態になったときに、評価をやめて終了します。

term　:　乗除算を評価します。term は、‘＊’、‘／’、‘¥’、‘%’以外の演算子が現れたときに終了します。

prime　:　べき乗を評価します。‘^’以外の演算子が現れたときに終了します。なお、べき乗の計算は実際には次のようにして行っていますので $x \leqq 0$ のときはエラーが起こります。

x ^ y　=　exp(ln(x) ＊ y)

factor　:　被演算子を評価します。被演算子は、各場合によって次のように処理されます。

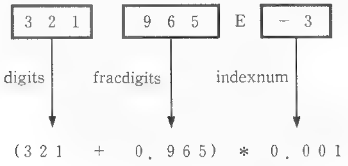
変数の場合：変数の値が格納されている配列(localvar)から、変数の値を取り出してきます。

数の場合　：ファンクション number に数式を評価させます。

関数の場合：ファンクション funs に関数を評価させます。

括弧の場合：ファンクション expression に括弧で括られた部分を評価させます。

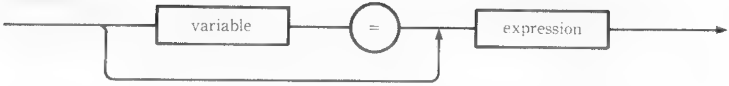
number　:　数を評価します。整数部分をファンクション digits に、小数部分をファンクション fracdigits、指数部分をファンクション indexnum に評価させています。また、１６進数の場合は、ファンクション hexdigits に評価させます。



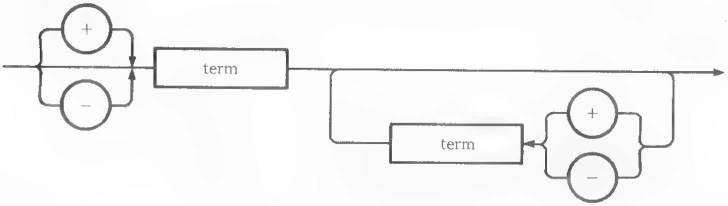
数の評価

funs　:　関数を評価します。関数への引数部分は、ファンクション expression に評価させます。関数の選択は、文字列中から関数名だけを取り出して、登録し

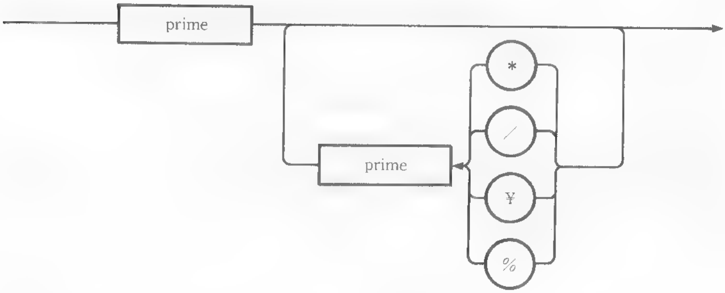
statement



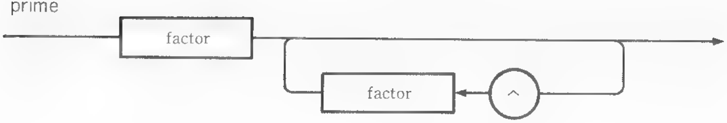
expression



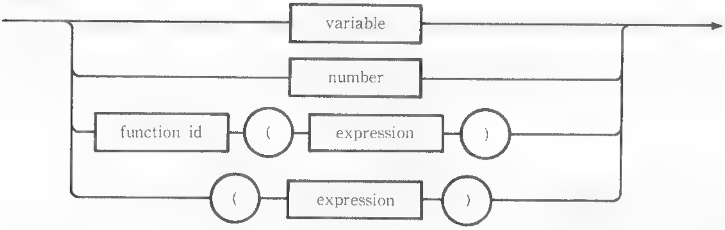
term

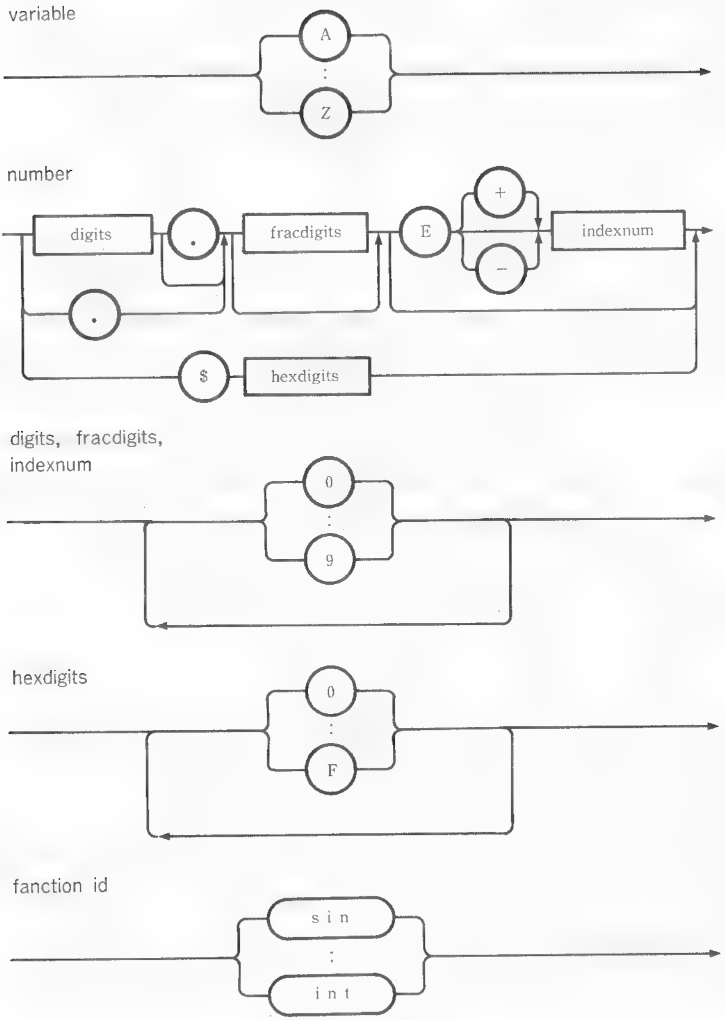


prime



factor





簡易計算機の構文グラフ

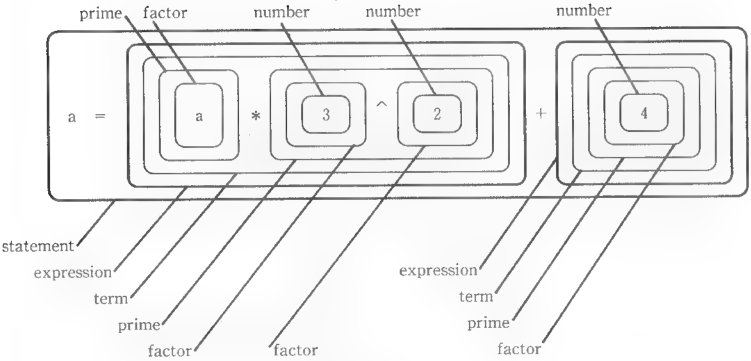
てある関数名と比較をすることによって行われます。

さて、実際に入力された文字列が評価されていく過程を追ってみましょう。例えば、次のように入力されたとします（変数ａの値は５６だとします）。

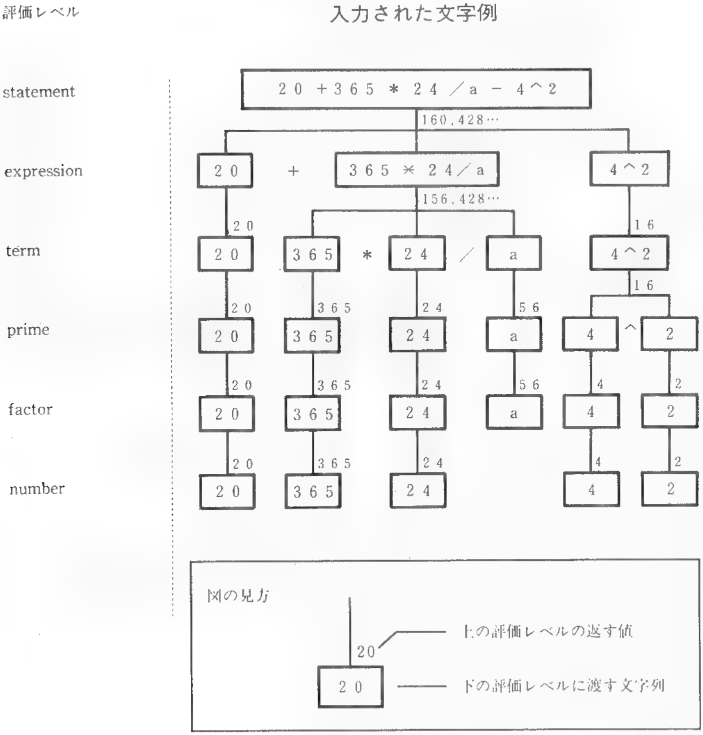
２０＋３６５＊２４／ａ－４＾２

この数式がstatementに渡されると、代入文でないのでそのままexpressionに受け渡されます。expressionでは、‘20’と‘365 ＊ 24/a’と‘4＾2’の部分に分け、各部分をtermに渡し、返ってきた値に対して加算と減算を行います。termでは３つの部分のうちの‘365 ＊ 24/a’の部分を、‘365’と‘24’と‘a’の部分に分け、その間で乗減算を行います。それぞれの部分はprimeで評価させます。primeでは、‘4＾2’の部分を‘4’と‘2’に分け、その間でべき乗の計算をします。分割された各部分はそのままfactorへ受け渡され、その値が評価されます。factorは、数をnumberに評価させ、‘a’に対する変数ａの値を配列から取り出してきます。numberでは、整数digitsに求めさせます(この例では、整数しかないので他のファンクションは呼び出しません)。この様子を図に表してみると次のようになります（statement～number間について示してあります)。

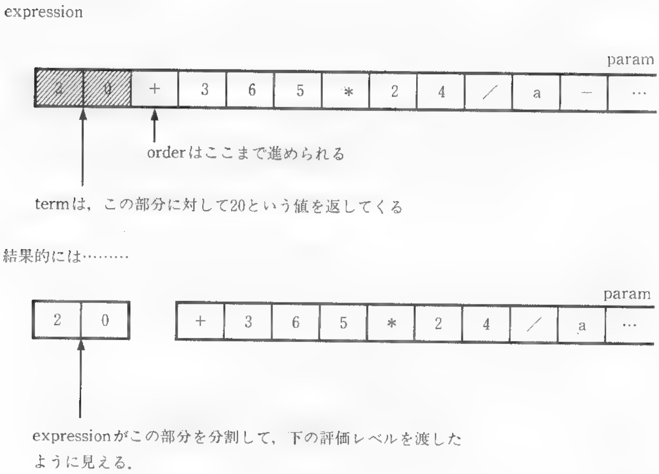
※プログラムでは、ある評価レベルで文字列を各部分に分けるのではなくて、下の評価レベルで評価された結果として、各部分に分かれたように見えるのです。例えば、expressionでは、まずtermを呼び出します。この結果、termは20という値を返してきます。また、文字列用のポインタも‘20’の後の位置を指すように更新されて返ってきます。だから、結果的にはexpressionにとってみれば、‘20’という文字列が加減演算子に対する被演算子という部分として分けられたことになるのです。expression自体が、文字列全体を検索して‘20’という部分を作り出し、それをtermに受け渡すということではありません。



数式の評価レベル



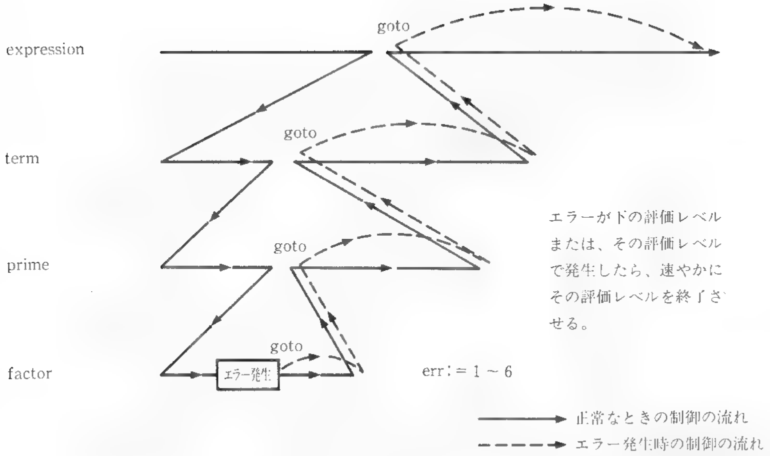
評価されている過程


expression
param
2
0
+
3
6
5
＊
2
4
／
a
−
…
orderはここまで進められる
termは，この部分に対して20という値を返してくる
結果的には………
param
2
0
+
3
6
5
＊
2
4
／
a
…
expressionがこの部分を分割して，下の評価レベルを渡したように見える．

エラーの処理は、Microware 社の Pascal では、1つのプロシジャの外側のステートメントに goto 文によって制御を移すことができないので、一か所でエラー処理をすることができません。ですから、ある評価レベルでエラーが起こったとき、エラー番号を変数 err にセットしてただちにその評価レベルを終了させます。その上の評価レベルでは、エラー（変数 err）をチェックし、エラー番号がセットして返されてきたときは、ただちにその評価レベルを終了させます。この操作の繰り返しによって、エラーが起こったときのアボート（Abort）処理を行っています。

出力の処理は、decimal（１０進数の出力）と hex（１６進数の出力）の２つのプロシジャで行っています。decimal では、答の値が 1.0E−10 より大きく、1.0E+10 より小さいときは、有効数字１０桁を出力するようにしています。その方法は、１０個の文字型の要素をもつ配列を用意しておいて、配列の後方から、整数部分の数字を入れていきます。そして、小数部分の数字を配列の先頭から、整数部分が入っている部分の手前まで詰めていきます。

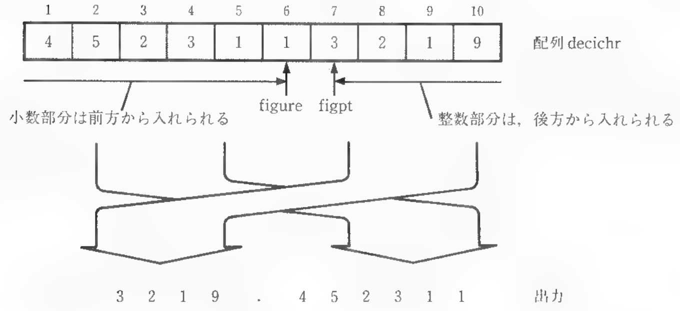
hex では、３２個の文字型の配列を用意しておきます。この計算機で扱える数の上限は 1.7E+38 ですから、16＾32＝3.403E+38 で３２桁あれば、数の整数部分を１６進数で表すことが可能になります。



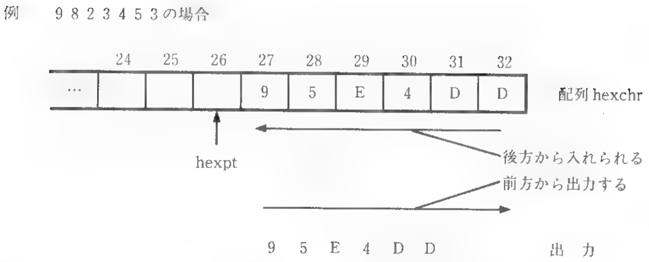
エラーが下の評価レベルまたは、その評価レベルで発生したら、速やかにその評価レベルを終了させる。

エラー発生時の処理

例　3219．4523112697 8の場合



decimalの動作例



hexの動作例

このプログラムでは、標準的なPascalの関数・プロシジャの他に、Microware社のPascal特有のものを用いています。それは、次のようなものです。

prompt　：　現在のテキストバッファをキャリジリターン、ラインフィードなしで、すぐに引き出すことができます（procedure）。

mathabort　：　引数としてFALSEを与えると、算術的なエラーによってプログラムを打ち切らないようにします（procedure）。

mathresult　：　算術的エラーが検知された最後のエラー番号を返します（function : integer）。

プログラムの記述は、ブロック構造は取らず、各ファンクション・プロシジャはメインの下にモジュールに分かれて並んでいます。また、数式に対するポインタorderと、エラー番号を持つ変数errは、グローバルにせずに、陽に各ファンクション・プロシジャに渡しています。また、インデント（字下げ）も、特別な方法をとっています。

```
program calculator (input,output);

const LENG     =250;
      NVAR     =26 ;
      NSYSPRM  =80 ;

      D_ZERO   =1  ;
      SYNTAX   =2  ;
      OVERFLOW =3  ;
      R_PARENT =4  ;
      FUNC_RNG =5  ;
      UND_FUNC =6  ;
```

```
label 10;

var  param    : array [0..LENG] of char;
     pointer  : integer;
     ch       : char ;
     localvar : array [1..NVAR] of real;


procedure initvar;

var  i:integer;
begin
     for i:=1 to NVAR do localvar[i]:=0.0
end; { of initvar }


procedure listvar;

var  i:integer;
begin
     for i:=1 to 13 do
     begin
          write   (chr(ord('A')-1+i),'=',localvar[i]);
          writeln ('    ',chr(ord('A')+12+i),'=',localvar[i+13])
     end;
     writeln
end; { of listvar }


function registvar (var err:integer; ch:char):integer;

var  num:integer;
begin
     if ord(ch)>=ord('a') then begin
          num:=ord(ch)-ord('a')+1;
          if num>NVAR then err:=SYNTAX  end
     else begin
          num:=ord(ch)-ord('A')+1;
          if num>NVAR then err:=SYNTAX
     end;
     registvar:=num
end; { of registvar }
procedure getch (var ch:char; var point:integer);
begin
     if point<=LENG then
          repeat
               ch    :=param[point];
               point:=point+1
          until (ch<>' ') or (point>LENG)
     else
          ch:=' '
end; { of getch }


function calcumode : boolean;

var  pointer:integer;
     switch :boolean;
begin
     pointer:=0;
     switch :=true;
     while (pointer<NSYSPRM) and switch do
     begin
          ch:=sysparam[pointer];
          if ch<>' ' then switch:=false;
          pointer:=pointer+1
     end;
     calcumode:=switch
end; { of calcumode }


procedure inputparam ;

var  i :integer;
     ch:char;
begin
```

```
     write('calcu:');
     for i:=0 to LENG do param[i]:=' ';
     i:=0;
     repeat
          prompt;
          read(ch);
          param[i]:=ch;
          i        :=i+1
     until eoln or (i>LENG)
end; { of inputparam }


procedure decimal (value:real);

const UPLEVEL =1.0E+10;
      LOWLEVEL=1.0E-10;
      DLENG   =10;

var   valreal , valfrac                 :real;
      valint  , figure , figpt , point :integer;
      decichr                           :array [1..DLENG] of char;
begin
     valreal:=abs(value);
     if (valreal<UPLEVEL) and (valreal>LOWLEVEL) then
     begin
          valfrac:=afrac(valreal);
          valreal:=aint (valreal);
          figpt  :=DLENG;
          while (valreal>0) and (figpt>0) do

          begin
               valint         :=trunc(valreal-aint(valreal/10)*10);
               decichr[figpt] :=chr(ord('0')+valint);
               valreal        :=aint(valreal/10.0);
               figpt          :=figpt-1
          end;
          figure:=figpt;
          figpt :=1;
          while (valfrac>0) and (figpt<=figure) do
          begin
               valfrac        :=valfrac*10.0;
               valint         :=trunc(valfrac);
               decichr[figpt] :=chr(ord('0')+valint);
               valfrac        :=afrac(valfrac);
               figpt          :=figpt+1
          end;
          if value<0.0 then write('-');
          if figure=DLENG then
               write('0')
          else
               for point:=figure+1 to DLENG do
                         write(decichr[point]);
          write('.');
          for point:=1 to figpt-1 do
                         write(decichr[point])
     end
     else
          if value=0.0 then write('0.0')
                       else write(value)
end; { of decimal }


procedure hex (value:real);

const HLENG = 32 ;

var  valreal        :real;
     valint , hexpt :integer;
     hexchr         :array [1..HLENG] of char;
begin
     valreal:=abs(aint(value));
     hexpt  :=HLENG;
     repeat
          valint :=round(valreal-(aint(valreal/16)*16));
          valreal:=aint(valreal/16);
          if valint<10 then
               hexchr[hexpt]:=chr(ord('0')+valint)
          else
               hexchr[hexpt]:=chr(ord('A')+valint-10);
          hexpt:=hexpt-1
     until (hexpt=0) or (valreal=0.0);
     if value<0.0 then write('-');
     repeat
          hexpt:=hexpt+1;
          write(hexchr[hexpt])
     until hexpt>=HLENG
end; { of hex }


function caldiv ( dividend, divisor :real ) : real ;

var  quotient  : real;
     minus     : boolean;
     quotient:=0;
     minus   :=false;
     if dividend<0.0 then  minus:=not minus ;
     if divisor <0.0 then  minus:=not minus ;
     dividend:=abs(dividend);
     divisor :=abs(divisor );
     while dividend >= divisor do
     begin
          dividend:=dividend-divisor;
          quotient:=quotient+1
     end;
     if minus then caldiv:= - quotient
              else caldiv:=   quotient
end; { of caldiv }


function calmod ( dividend, divisor : real ) :real ;

var  minus : boolean ;
begin
      minus:=false;
      if dividend<0.0 then  minus:=not minus;
      if divisor <0.0 then  minus:=not minus;
      dividend:=abs(dividend);
      divisor :=abs(divisor );
      while dividend >= divisor do
            dividend := dividend - divisor;
      if minus then calmod:=-dividend
               else calmod:= dividend
end; { of calmod }


function digits (var order:integer):real;

var  pointer   : integer;
     ch        : char;
     temp ,ans : real;
begin
     pointer:=order;  getch(ch,pointer);
     ans:=0.0;
     while ch in ['0'..'9'] do
     begin
          getch (ch,order);
          temp   :=ord(ch)-ord('0');
          ans    :=ans*10.0+temp;
          getch (ch,pointer)
     end;
     digits:=ans
end; { of digits }


function fracdigits (var order:integer):real;

var  ans , temp ,devider :real;
     pointer             :integer;
     ch                  :char;
begin
     pointer:=order;  getch(ch,pointer);
     ans    :=0.0;
     devider:=1.0;
```

```
          while ch in ['0'..'9'] do
          begin
               getch (ch,order);
               temp    :=ord(ch)-ord('0');
               ans     :=ans*10.0+temp;
               devider:=devider*10.0;
               getch (ch,pointer)
          end;
          fracdigits:=ans/devider;
     end; { of fracdigits }


     function hexdigits (var order:integer):real;

     var  pointer     :integer;
          ch          :char;
          ans , temp  :real;
     begin
          pointer:=order;  getch(ch,pointer);
          ans:=0.0;
          while ch in ['0'..'9','a'..'f','A'..'F'] do
          begin
               order:=pointer;
               if ch in ['0'..'9'] then
                    temp:=ord(ch)-ord('0')
               else
                    if ch in ['a'..'f'] then
                         temp:=ord(ch)-ord('a')+10.0
                    else
                         temp:=ord(ch)-ord('A')+10.0;
               ans:=ans*16.0+temp;
               getch(ch,pointer)
          end;
          hexdigits:=ans
     end; { of hexdigits }


     function indexnum ( var err,order :integer ):real;

     var  pointer    : integer;
          ch         : char;
          temp       : real;
          minus      : boolean;
     begin
          pointer:=order;  getch(ch,pointer);
          minus  :=false;
          if ch in ['+','-'] then
          begin
               order:=pointer;
               if ch='-' then minus:=true;
               getch(ch,pointer)
          end;
          if ch in ['0'..'9'] then
          begin
               temp:=digits(order);
               if minus then
                    indexnum:=exp(ln(10.0)*(-temp))
               else
                    indexnum:=exp(ln(10.0)*temp)
          end
          else
          begin
               err      :=SYNTAX;
               indexnum:=0.0
          end;
          if err=0 then err:=mathresult
     end; { of indexnum }


     function number (var err,order:integer):real;

     var  pointer    : integer;
          ch         : char;
          temp ,ans  : real;
     begin
          pointer:=order;  getch(ch,pointer);
          ans :=0.0;
     if ch='$' then
     begin
          order:=pointer;
          ans  :=hexdigits(order)
     end
     else
     begin
          if ch in ['0'..'9'] then ans:=digits(order);
          pointer:=order;
          getch(ch,pointer);
          if ch='.' then
          begin
               order:=pointer;
               getch(ch,pointer);
               if ch in ['0'..'9'] then
               begin
                    ans     :=ans+fracdigits(order);
                    pointer:=order;
                    getch(ch,pointer)
               end
          end;
          if (ch='E') or (ch='e') then
          begin
               order:=pointer;
               ans  :=ans * indexnum(err,order)
          end
     end;
     number:=ans;
     if err=0 then err:=mathresult
end; { of number }


function expression(var err,order:integer):real;forward;


function funs (var err,order:integer):real;

const FLENG = 7 ;

var  pointer , namept :integer;
     ch               :char;
     ans              :real;
     funcid           :array [1..FLENG] of char;
label 1;
begin
     for namept:=1 to FLENG do funcid[namept]:=' ';
     namept:=1;
     getch(ch,order);
     while (ch<>'(') and (namept<8) do
     begin
          if ch in ['a'..'z'] then ch:=chr(ord(ch)-ord('a')+ord('A'));
          funcid[namept]:=ch;
          namept:=namept+1;
          getch(ch,order)
     end;
     if ch<>'(' then begin err:=UND_FUNC ;goto 1 end;
     ans:=expression(err,order);
     if err<>0 then goto 1;
     getch(ch,order);
     if ch<>')' then begin err:=R_PARENT ;goto 1 end;
   {case statement can't be used}
   if funcid='SIN    ' then ans:=sin(ans) else
   if funcid='COS    ' then ans:=cos(ans) else
   if funcid='TAN    ' then ans:=sin(ans)/cos(ans) else
   if funcid='ARCSIN ' then ans:=arctan(ans/sqrt(1-sqr(ans))) else
   if funcid='ARCCOS ' then ans:=1.570796327-arctan(ans/sqrt(1-sqr(ans))) else
   if funcid='ARCTAN ' then ans:=arctan(ans) else
   if funcid='SINH   ' then ans:=(exp(ans)-exp(-ans))/2 else
   if funcid='COSH   ' then ans:=(exp(ans)+exp(-ans))/2 else
   if funcid='TANH   ' then ans:=(exp(ans)-exp(-ans))/(exp(ans)+exp(-ans)) else
   if funcid='ARCSINH' then ans:=ln(ans+sqrt(sqr(ans)+1)) else
   if funcid='ARCCOSH' then ans:=ln(ans+sqrt(sqr(ans)-1)) else
   if funcid='ARCTANH' then ans:=ln((1+ans)/(1-ans))/2 else
   if funcid='SEC    ' then ans:=1/cos(ans) else
   if funcid='CSC    ' then ans:=1/sin(ans) else
   if funcid='COT    ' then ans:=cos(ans)/sin(ans) else
   if funcid='ARCSEC ' then begin
```

```
                              if ans>0.0 then
                                   ans:=arctan(sqrt(sqr(ans)-1))
                              else
                                   ans:=arctan(sqrt(sqr(ans)-1))-3.14159265359
                              end else
     if funcid='ARCCSC ' then begin
                              if ans>0.0 then
                                   ans:=arctan(1/sqrt(sqr(ans)-1))
                              else
                                   ans:=arctan(1/sqrt(sqr(ans)-1))-3.1415926536
                              end else
     if funcid='ARCCOT ' then ans:=1.570796327-arctan(ans) else
     if funcid='SECH   ' then ans:=2/(exp(ans)+exp(-ans))  else
     if funcid='CSCH   ' then ans:=2/(exp(ans)-exp(-ans))  else
     if funcid='COTH   ' then ans:=(exp(ans)+exp(-ans))/(exp(ans)-exp(-ans)) else
     if funcid='ARCSECH' then ans:=ln((1+sqrt(1-sqr(ans)))/ans) else
     if funcid='ARCCSCH' then begin
                              if ans>0.0 then
                                   ans:=ln((1+sqrt(1+sqr(ans)))/ans) else
                              if ans=0.0 then
                                   ans:=ln(1/ans)
                              else
                                   ans:=ln((1-sqrt(1+sqr(ans)))/ans)
                              end else
     if funcid='ARCCOTH' then ans:=ln((ans+1)/(ans-1))/2 else
     if funcid='EXP    ' then ans:=exp(ans) else
     if funcid='LOG    ' then ans:=ln(ans)/ln(10.0) else
     if funcid='LN     ' then ans:=ln(ans) else
     if funcid='SQR    ' then ans:=sqr(ans) else
     if funcid='SQRT   ' then ans:=sqrt(ans) else
     if funcid='RAD    ' then ans:=ans*3.1415926536/180.0 else
     if funcid='DEG    ' then ans:=ans/3.1415926536*180.0 else
     if funcid='INT    ' then ans:=aint(ans)
     else err:=UND_FUNC;
1:   if ( err=0 ) and ( mathresult<>0 ) then err:=FUNC_RNG;
     funs:=ans
end; { of funs }


function factor(var err,order:integer):real;

var  pointer:integer;
     ans    :real;
     ch     :char;
label 2;
begin
     pointer:=order;
     getch(ch,pointer);
     if ch in ['0'..'9','.','$'] then
          ans:=number(err,order)
     else
          if ch in ['a'..'z','A'..'Z'] then
          begin
               getch(ch,pointer);
               if ch in ['a'..'z','A'..'Z','('] then
                    ans:=funs(err,order)
               else
               begin
                    getch(ch,order);
                    if ch in ['a'..'z'] then
                         ans:=localvar[ord(ch)-ord('a')+1]
                    else
                         ans:=localvar[ord(ch)-ord('A')+1]
               end
          end
          else
               if ch='(' then
               begin
                    order:=pointer;
                    ans  :=expression(err,order);
                    if err<>0 then goto 2;
                    getch(ch,order);
                    if ch<>')' then err:=R_PARENT
               end
               else
                    err:=SYNTAX;
```

```
2:   factor:=ans
end; { of factor }


function prime (var err,order:integer):real;

var  pointer     : integer;
     ch          : char;
     ans , temp : real;
label 3;
begin
     ans:=factor(err,order);
     if err<>0 then goto 3;
     pointer:=order;
     getch (ch,pointer);
     while ch='^' do
     begin
          order   :=pointer;
          temp    :=factor(err,order);
          if err<>0 then goto 3;
          ans     :=exp(ln(ans)*temp);
          pointer:=order;
          getch (ch,pointer)
     end;
     err  :=mathresult;
3:   prime:=ans
end; { of prime }


function term (var err,order:integer):real;

var  pointer     : integer;
     ch          : char;
     ans , temp : real ;
label 4;
begin
     ans:=prime(err,order);
     If err<>0 then goto 4;
     pointer:=order;
     getch (ch,pointer);
     while ch in ['*','/','¥','%'] do
     begin
          order:= pointer;
          temp := prime(err,order);
          if err<>0 then goto 4;
          if ch = '*' then
               ans:=ans * temp
          else
          begin
               if temp=0.0 then
               begin
                    err:=D_ZERO;  goto 4;
               end;
               case ch of
                    '/' : ans:=ans / temp ;
                    '¥' : ans:=caldiv(ans,temp) ;
                    '%' : ans:=calmod(ans,temp)
               end
          end;
          pointer:=order;  getch(ch,pointer)
     end;
     err:=mathresult;
4:   term:=ans
end; { of term }


function expression; { (var err,order : integer) : real }

var  pointer: integer;
     ch     : char;
     ans    : real;
     first  : boolean;
label 5 ;
begin
     pointer:=order;  getch(ch,pointer);
     ans     :=0.0;
     first  :=true;
```

```
      while not (ch in [' ',')']) do begin
           if ch in ['+','-'] then
           begin
                order:=pointer;
                case ch of
                     '+' : ans:=ans+term(err,order);
                     '-' : ans:=ans-term(err,order)
                end;
                if err<>0 then goto 5
           end
           else
                if first then
                begin
                     ans  :=term(err,order);
                     if err<>0 then goto 5
                end
                else
                begin
                     err:=SYNTAX;  goto 5
                end;
           if first then first:=false;
           pointer:=order;
           getch(ch,pointer);
      end;
      if first and (ch=')') then  err:=SYNTAX
                            else  err:=mathresult;
 5:   expression:=ans
 end; { of expression }

procedure statement;

var  pointer, numvar, order, err : integer ;
     answer                      : real ;
     ch                          : char ;
begin
     order  :=0;
     err    :=0;
     pointer:=order;
     getch(ch,pointer);  getch(ch,pointer);
     if ch='=' then
     begin
          getch(ch,order);
          numvar:=registvar(err,ch);
          if err=0 then
          begin
               order:=pointer;
               answer          :=expression(err,order);
               localvar[numvar]:=answer
          end
     end
     else
          answer:=expression(err,order);
     if err=0 then
     begin
          write ('answer=' );  decimal(answer);
          write ('    hex=');  hex    (answer);
          writeln;  writeln
     end
     else
     begin
          if err>UND_FUNC then err:=OVERFLOW;
          write('error ',err:1,' occured at ');
          if order>LENG then writeln('last columns')
                        else writeln(order:3,' columns');
          writeln
     end
end; { of statement }


begin { main }
     mathabort(false);
     initvar;
     if not calcumode then
          for pointer:=0 to LENG do
               if pointer<NSYSPRM then  param[pointer]:=sysparam[pointer]
                                  else  param[pointer]:=' '
     else
```

```
        begin
             writeln;  writeln('calculate mode');
             inputparam;
        end;
        while ( not eof ) or ( not calcumode ) do
        begin
             pointer:=0;
             getch(ch,pointer);
             if ch='?' then  listvar
                       else  statement;
             if not calcumode then goto 10;
             inputparam
        end;
   10:  writeln
   end. { of main }
```

## ２－２－１３　cal

西暦１年～１００００年の間のカレンダーを標準出力に出力します。

指定した年のカレンダーを見たい場合は次のように入力します。

　　ｃａｌ　年

例　ｃａｌ　１９８４

指定した月のカレンダーを見たい場合は次のように入力します。

　　ｃａｌ　年　月

例　ｃａｌ　１９９９　７

mainプログラムは、コマンドのパラメータが2個ある場合と1個ある場合とに分け、2個の場合はmonthcalを、1個の場合はyearcalを呼び出します。それぞれのサブルーチンには、パラメータ（年、または年と月）を整数に変換したものを渡します。

monthcalは、ひと月の表示を行います。まず、与えられた年と月の日数とその月の第1日が始まる曜日を、monthdayとstartdayという2つの関数を呼び出すことによって、求めています。第1日が始まる曜日は0～6（日曜を0とし土曜を6とする）で返されますので、それをオフセットとして変数dayoftに記憶しておきます。その月のある日(dayと置きます)を出力して、day＋dayoftが7の倍

数になったときは改行します。例えば、出力する月が土曜から始まるときは、dayoft が 6 になります。第 1 日（day＝1）を出力したあとは、day＋dayoft が 6＋1＝7 で 7 の倍数になりますから、改行します。このようにして、その月の日数分（変数 numday が日数を記憶している）だけ、日を出力していきます。

yearcal は、1 年の表示を行うため、1 行に 3 つの月のカレンダーを出力しなければなりません。ですから、monthcal で使用していた変数をサイズ 3 の配列として用意します。monthcal で述べた day＋dayoft が 7 の倍数になったときは、改行するのではなくて次の月の出力を行います。3 ケ月分出力したときに始めて改行するのです。この動作を 6 行分行います（ひと月は、最高でも 6 行あれば表示できるからです）。ただし、6 行出力する間に、もうその月の日数分出力してしまったら、日を出力するかわりに空白を出力します。

monthday は、指定された月の日数を求めます。演算によって月の日数を求める方法もあるのですが、ここでは１２月分の日数のデータをサイズ１３の配列にとっています（配列は 0 から始まるためにサイズを１３にしています）。ただし、次の条件を満たす年は閏年になりますから、2 月を指定した場合は＋１します。

閏年の条件：その年が 4 の倍数であり、かつ１００の倍数でないこと。または
　　　　　　その年が４００の倍数であること。

startday は、指定された月の第 1 日が始まる曜日を 0 ～ 6 までの整数の年で求めます。配列 start_day には、1 月 1 日が日曜であるときのそれぞれの月の第 1 日が始まる曜日が整数のデータとして入れてあります（閏年でないときのものです）。

ここで、ある年の 1 月 1 日の曜日を求めてみます（曜日は日曜を 0、土曜を 6 とします）。1 年を３６５日とすると、365 mod 7 ＝ 1 ですから、ある年の翌年の 1 月 1 日は、その年の 1 月 1 日が始まる曜日の次の曜日になります。例えば、ある年の 1 月 1 日が火曜日だったとすると、その翌年の 1 月 1 日は水曜日になります。閏年がないかぎり、毎年この方法で計算できますから、西暦 x 年の 1 月 1 日は、(x－1＋1) mod 7 で計算できます（－１しているのは、西暦が 1 年から始まるからで、西暦 1 年との差を求めているのです。＋１しているのは、西暦 1 年の 1 月 1 日が月曜から始まっているからです）。閏年が入ることを考えれば、x に西暦 1 年からその年の前年までの間の閏年の入った分だけ＋１してやらなければ

なりません。西暦x年までの閏年の回数はx−1をyと置くと次のように計算できます。

$$(y \quad \mathrm{div} \quad 4)-(y \quad \mathrm{div} \quad 100)+(y \quad \mathrm{div} \quad 400)$$

よって、西暦x年の1月1日が始まる曜日は、x−1をyと置くと次のようになります。

$$(y+(y \quad \mathrm{div} \quad 4)-(y \quad \mathrm{div} \quad 100)+(y \quad \mathrm{div} \quad 400)+1) \quad \mathrm{mod} \quad 7$$

今まで1月1日について扱ってきましたが、同じことが1年の他のすべての日の曜日についてもあてはまります。ただし、2月28日より後は、その年が閏年なら曜日が1つ後ろへずれます。

同じようにして、ある月の第1日の曜日をプログラムのように求めています(プログラムでは7の剰余計算を行う前に、start_day［指定した月］の値と閏年ならそれに＋1を足しています。

※文章中で用いた記号は、modは整数剰余を、divは整数除算を示します。

```
/*    cal  --  calender    */

/*    usage: cal year <month>.    */


#include <stdio.h>

#define  isleap( year ) ((year%4 == 0) && (year%100 != 0) || (year%400 == 0))

char *mname[13] = {
    "", "Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};
char week[] = " Su Mo Tu We Th Fr Sa";

main( argc, argv )
     int   argc;
     char *argv[];

{
     int  year, month;

     if ( argc == 3 ) {
          year = atoi( argv[1] );
          month = atoi( argv[2] );
          if ( month > 12 || month < 1 || year < 1 || year > 10000 ) {
               fprintf( stderr, "I don't know such ■ year or month !¥n" );
               return;
          }
          monthcal( year, month );
     } else if ( argc == 2 ) {
          year = atoi( argv[1] );
          if ( year > 0 && year < 10001 )
                yearcal( year );
     }
}


monthcal( year, month )
     int  year, month;
{
     int  numday, dayoft, day;

     printf( "        %s %d¥n¥n%s¥n", mname[month], year, week );
     numday = monthday( year, month );
     dayoft = startday( year, month );

     space( dayoft*3 );
     for ( day = 1; day <= numday; day++ ) {
          printf( "%3d", day );
          if ( ( day + dayoft ) % 7 == 0 )
               putchar( '¥n' );
     }
     if ( ( numday + dayoft ) % 7 != 0 )
          putchar( '¥n');

}
```

```
          if ( ( day + dayoft ) % 7 == 0 )
               putchar( '¥n' );
     }
     if ( ( numday + dayoft ) % 7 != 0 )
          putchar( '¥n');
```

```
}


yearcal( year )
     int  year;
{
     int  numday[3], dayoft[3], day[3], i, month, moft;

     space( 26 );
     printf( "calendar of %d¥n¥n", year );
     for ( moft = 1; moft < 12; moft += 3 ) {
          for ( month = 0; month < 3; month++ ) {
               numday[month] = monthday( year, moft+month );
               dayoft[month] = startday( year, moft+month );
               space( month? 20: 9 );
               printf( mname[ moft+month ] );
          }
          printf( "¥n¥n" );
          for ( month = 0; month < 3; month++ )
               printf( "%s  ", week );
          putchar( '¥n' );

          day[0] = day[1] = day[2] = 1;
          for ( i = 0; i < 6; i++ ) {
               for ( month = 0; month < 3; month++ ) {
                    space( ( day[month] == 1 )? 3*dayoft[month]: 0 );
                    do {
                         if ( day[month] < numday[month]+1 )
                              printf( "%3d", day[month] );
                         else
                              printf( "   " );
                    } while ( (day[month]++ + dayoft[month]) % 7 != 0 );
                    printf( "  " );
               }
               putchar( '¥n' );
          }
          printf( "¥n¥n" );
     }
}


monthday( year, month )
     int  year, month;
{
     static char month_day[13] = {
          0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
     };
```

```
        if ( month == 2 && isleap( year ) )
                return( month_day[month] + 1 );
        else
                return( month_day[month] );
}


startday( year, month )
        int  year, month;
{
        int  sday;
        static char start_day[13] = {
                0, 0, 3, 3, 6, 1, 4, 6, 2, 5, 0, 3, 5
        };
        year--;
        sday = year + (year/4) - (year/100) + (year/400) + start_day[month] + 1;
        if ( month > 2 && isleap( ++year ) )
                return( (sday+1) % 7 );
        else
                return( sday % 7 );
}


space( tab )
        int  tab;
{
        int i;

        for ( i = 1; i <= tab; i++ )
                putchar( ' ' );
}
```

## 2—2—14　lprint

指定したファイルの特定の単語を強調印字で出力するよう制御するプログラムを作成しました。指定する単語は、最高１００語まで登録することが可能です.

使用方法は、次の様にコマンド入力します。

lprint　登録ファイル　入力ファイル〈コード１　コード２　コード３〉

登録ファイルとは、強調印字を指定する単語が書かれているファイルのことです。登録ファイル中で、各単語は、改行か空白で区切られていなければなりません。１つの単語の長さは、最高20文字まで、単語の数は最高１００語まで登録することができます。

入力ファイルとは、強調印字する前の元のテキストが書かれているファイルのことです。入力ファイルでは強調印字される単語は、空白か改行で区切られていることが必要です。ですから、一続きの単語の一部分が指定した単語の綴と同じときでも、その部分だけ強調印字をするということはありません。例えば、

We　want　to　use　Pascal　language.

という文章が入力ファイルにあったとき、登録ファイル中にlanguageという単語が指定されていても、language.は、ピリオドが後にあるため強調印字されません。この場合、ピリオドとの間に１つ以上の空白を空けることが必要となります。また、大文字と小文字の区別も行います。

コードとは、強調印字を指定・解除するコードのことで２桁以内の16進法で指定します。ある単語を強調印字するときは、単語を出力する前に次のように出力されます。

コード１　＋　コード２　（省略時は、ESC＋E　(1b＋45)　）

単語を出力し終った後は、次のように出力されます。

コード１　＋　コード３　（省略時は、ESC＋F　(1b＋46)　）

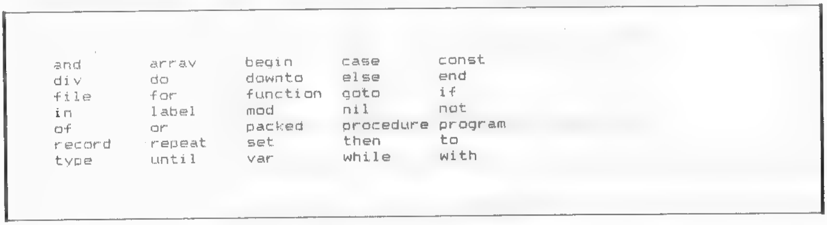
これは、各プリンタによって強調印字するシーケンスコードが異なるために、ユーザが自分のプリンタに適合したシーケンスを出力できるようにという目的で組み込んだものです。なお、省略時にはESC+EとESC+Fが用いられます。

出力は、標準出力に対して行われます。ですから、プリンタへ出力したい場合は、次のように、リダイレクトで出力をプリンタに割り当てなければなりません。

```
lprint  registfile  infile  1b  49  58  >/p
```

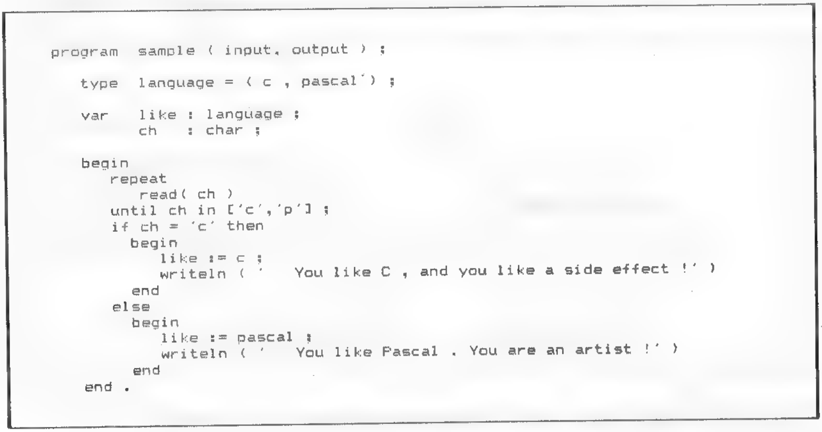
例えば、Pascalのソースファイルで、予約語だけを強調印字したい場合を考えてみましょう。

ファイル"reserved"は、次のように作成します。



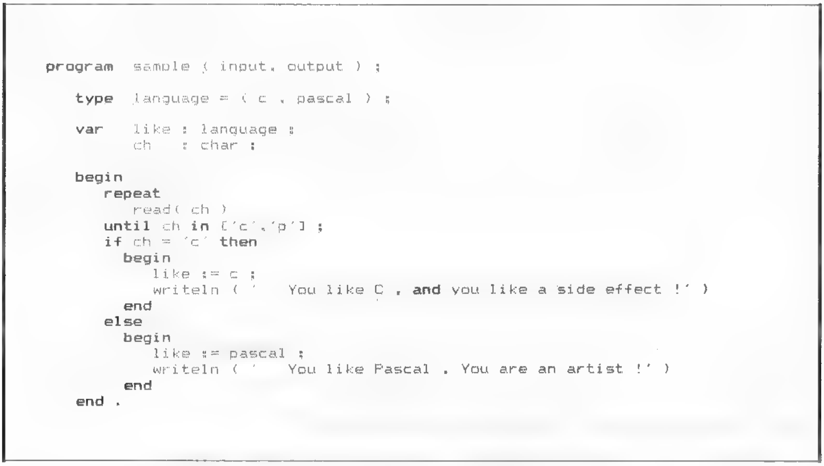
```
and       array     begin     case      const
div       do        downto    else      end
file      for       function  goto      if
in        label     mod       nil       not
of        or        packed    procedure program
record    repeat    set       then      to
type      until     var       while     with
```

また、ファイル"sample. p"は、次のように書かれていたとします.



```
program  sample ( input, output ) ;

   type  language = ( c , pascal ) ;

   var   like : language ;
         ch   : char ;

   begin
      repeat
         read( ch )
      until ch in ['c','p'] ;
      if ch = 'c' then
        begin
           like := c ;
           writeln ( '   You like C , and you like a side effect !' )
        end
      else
        begin
           like := pascal ;
           writeln ( '   You like Pascal . You are an artist !' )
        end
   end .
```

このとき、lprint reserved sample. p >/pを実行します。



```
program  sample ( input, output ) ;

   type  language = ( c , pascal ) ;

   var   like : language ;
         ch   : char ;

   begin
      repeat
         read( ch )
      until ch in ['c','p'] ;
      if ch = 'c' then
        begin
           like := c ;
           writeln ( '    You like C , and you like a side effect !' )
        end
      else
        begin
           like := pascal ;
           writeln ( '    You like Pascal . You are an artist !' )
        end
   end .
```

プログラムの構成は、次のようになっています。

mainプログラムは、argcの数を見て引数の数がおかしい場合は、使用法の説明を、usageサブルーチンを用いて、標準エラー出力へ出力するようにしています。引数が、コマンドも含めて3個の場合は、コード1～3（プログラム中では、コード1をseg、コード2をeph、コード3をcanという名の変数で扱っています）にデフォルトの値を渡します。引数が6個のときは、16進数文字列を整数に変換して、各々のコードを示す変数に代入しています。そして、gsymを呼び出します。

gsymサブルーチンは、登録ファイルから単語をfscanfを使って拾ってきては、nameという名前の配列に格納していきます。gsymは、単語をいくつ登録したかというカウンタcntを持っており、cntの最終値をmainプログラムに返してきます。

mainプログラムは、gsymで返ってきた単語の数を検討し、もし0なら何かの異常が起こったと判断し、エラーメッセージを出力して終了します。そうでないかぎり、その単語の数と、先程求めた3つのコードなどをthroughに受け渡して、終了します。

throughサブルーチンは、入力ファイルから、getcを用いて、1文字づつ入力してきます。入力されたキャラクタは、wordという名前の配列に格納されていきます(wordは、NULLによって初期化されています)。入力してきた文字が、EOF（ファイルの終了）かEOL（行の終了）かSPC（空白）でない限り、wordに文字を格納していきますが、20文字以上になった場合は、そこで格納をするのを中止します。文字を1文字もwordに格納していないうちに、EOFやEOLやSPCがきた場合は、EOFのときは終了し、EOLやSPCのときはそれらの文字を直接出力します。単語の最後にそれらの文字がきたときは、次の入力でそれらの文字が検索されるように、ungetc( )を用いてバッファに戻してやります。wordに何文字か入力された場合は、gsymで登録した単語の中に、それと一致するものがあるか調べます。もし、一致した場合は、強調印字コード・単語・解除コードの順で、その単語を出力します。一致しないときは、その単語をそのまま出力します。この動作を、先程述べたEOFがくるまで繰り返しています。

コード1～3は、各プリンタによって変わります。デフォルトとして入っているのは、富士通のシリアルドットプリンタに対するものです。ビジネスプリンタ（富士通製）に対しては、次のようにコードを指定します（ただし、ビジネスプリンタは強調印字モードはなく、アンダーラインで代用しています)、その他のプリンタについては、お手持ちのマニュアルで調べて設定、改良してください。

| | コード1 | コード2 | コード3 |
|---|---|---|---|
| ・シリアルプリンタ<br>(デフォルト値) | 1b　(ESC) | 45　('E') | 46　('F') |
| ・ビジネスプリンタ<br>(アンダーライン) | 1b　(ESC) | 49　('I') | 58　('X') |

コードを変えて、自分の指定したファイルへ書き込むようにすれば、別な用途にも使えます。例えば、コード1を2a、コード2とコード3を2aとすれば、出力ファイルには、登録された単語の前後にはアスタリスクが2つつくことになります。

プログラムは、単語登録時に辞書順に単語をソートしておけば、検索するときに二分検索法が使えるのでより速くなります。登録単語数をもっと増やして使用するときは、そのようにしてください。

```
#include  <stdio.h>

#define   ESCC  0x1b
#define   EPHC  0x45
#define   CANC  0x46
#define   SPC   0x20
#define   NO    0

#define   WMAX  100
#define   WLEN  20


main( argc, argv )
        int   argc;
        char *argv[];
{
        int   n, gsym(), seq, eph, can;

        if ( argc != 3 && argc != 6 ) {
                usage();
                return;
        }

        if ( argc == 3 ) {
                seq = ESCC;
                eph = EPHC;
                can = CANC;
        } else {
                seq = htoi( argv[3] );
                eph = htoi( argv[4] );
                can = htoi( argv[5] );
        }

        if ( ( n = gsym( argv[1] ) ) == NO ) {
                fprintf( stderr, "registration error¥n" );
                return;
        }
        through( argv[2], n, seq, eph, can );
}


usage()
{
        fprintf( stderr, "usage incorrect¥n" );
        fprintf( stderr, "  lprint  registfile  inputfile  < seq eph can > ¥n" );
        fprintf( stderr, "       seq - sequence hex code ¥n" );
        fprintf( stderr, "       eph - emphasis hex code ( seq + eph )¥n" );
        fprintf( stderr, "       can - cancell  hex code ( seq + can )¥n" );
        fprintf( stderr, "       default : seq = 1b  eph = 45  can = 46¥n¥n" );
}


char name[WMAX][WLEN];

int   gsym( fname )
        char *fname;
{
        FILE *fp, *fopen();
        int   cnt;

        if ( ( fp = fopen( fname, "r" ) ) == NULL )
                return( NO );

        for ( cnt = 0; cnt < WMAX; cnt++ )
                if ( fscanf( fp, "%19s", name[cnt] ) == EOF )
                        break;
```

```
    fclose( fp );
    return( cnt );
}


#define testc( fp )  ( ( c = getc( fp ) ) == EOF || c == EOL || c == SPC )


through( fname, n, seq, eph, can )
    char *fname;
    int  n, seq, eph, can;
{
    FILE *fp, *fopen();
    int  getc(), c, i, j;
    char word[WLEN];

    if ( ( fp = fopen( fname,"r" ) ) == NULL ) {
        fprintf( stderr, "can't open input file¥n" );
        return;
    }

    for ( ; ; ) {
        for ( i = 0; i < WLEN; i++ )
            word[i] = NULL;

        if ( !testc( fp ) )
            for ( i = 0; i < WLEN; i++ ) {
                word[i] = c ;
                if ( testc( fp ) ) {
                    ungetc( c, fp );
                    break;
                }
            }

        if ( strlen( word ) == 0 ) {
            if ( c == EOF ) {
                fclose( fp );
                return;
            } else if ( c == EOL || c == SPC )
                putc( c, stdout );
        } else {
            for ( j = 0; j < n ; j++ )
                if ( strcmp( word, name[j] ) == 0 )
                    break;
            if ( j >= n )
                fputs( word, stdout );
            else {
                putc( seq, stdout );
                putc( eph, stdout );
                fputs( word, stdout );
                putc( seq, stdout );
                putc( can, stdout );
            }
        }
    }
}


#define   MASK 0xfff0

int htoi( p )
    char *p;
{
    int h, c;

    for ( h = 0; c = *p; p++ )
        if ( c >= '0' && c <= '9' )
            h = ( ( h << 4 ) & MASK ) + c - '0';
        else if ( ( c |= 0x20 ) >= 'a' && c <= 'f' )
            h = ( ( h << 4 ) & MASK ) + c - 'a' + 10;
        else
            break;
    return( h );
}
```

## 2－2－15　grep

grepとはようするにサーチのことです。サーチの文字列を指定するのに正規表現という一種の文法規則のようなものを使用します。ワイルドカードはその変形と考えればよいでしょう。UNIXでも正規表現のすべてをサポートするものは少ないようですし、特にOS—9では使用できるキャラクタに制限がありますので、次のものだけを採用します。

| | |
|---|---|
| * | 任意長の文字列 |
| ? | 任意の１文字 |
| ^a | aでない１文字 |
| % | 行の先頭 |
| $ | 行の最後 |
| ¥ | エスケープ文字（¥の直後の文字そのもの） |
| ¥１b | ¥の直後の２文字を16進数とする文字 |
| [abc] | aまたはbまたはcの１文字 |
| [^abc] | aでなくbでなくcでない１文字 |
| a—x | ＡＳＣＩＩコードでaからxまでの文字<br>ただし［　］の中でのみ |

‘*’は、0文字以上の任意の文字列を表すことに注意して下さい。その前に何があるかには関係ありません。またgrepはパターンによるサーチですから入力した文字列の中にパターンが表す文字列が含まれていればよく、ワイルドカードの場合のように両端が一致する必要がありません。あえて両端を合わせたい場合には‘%’および‘$’のメタキャラクタを使います。

grepの実現方法としてはまず「ソフトウェア作法」によるものがあります。これはコマンドラインで指定したパターンをいったん後で処理しやすいように整形するというもので、最も一般的と思われるのでここでもそれに倣いました。ただし、整形の仕方、および整形したパターンの格納パターンの格納場所については違った方法をとります。

まず、整形はもともとのパターン文字列に直接施します。別の配列にコピーしたりしません。こうする理由は、コマンドライン上に２つ以上のパターンを置くことを許したいからですが、詳しいことは後でのべます。

文字列そのものに手を加えるということにすると、後の設計方針が決まってきます。何より、整形した結果がもとより大きくなってはいけません。メタキャラクタの位置は、それを表す（そして通常のキャラクタと区別のできる）１バイトのキャラクタで置き替えてしまいます。

問題は、１バイトで通常のキャラクタと区別できるものがあるかということですが、そんなものが有り得るわけがありません。結論から言いますとＡＳＣＩＩ文字セットのうち８個ほどをメタキャラクタ用に特別視することになります。それは普段めったに使われることがなく、しかもシフトＪＩＳコードともかち合わないものが望ましいというところから、＄Ｆ1から＄Ｆ8までを使用することにしました。したがってこれらのキャラクタそのものをサーチすることは絶対にできません。ただし、何を使うかはマクロで定義していますので、変更してしまってもかまいません。かえってコントロールコード（＄01から＄1F）にした方がよいかもしれません。

さてgrepもまたデバッグの都合上、分割コンパイルにしました。最初に挙げるのは、パターンを整形する関数makepatです。

makepat自身は、パターンに間違いが無ければＹＥＳを返し、間違いがあればＮＯを返すことにします。ｐで表される文字列を１バイトずつ追いかけてゆき、メタキャラクタが現れたならばその位置を記号で置き換えます。ただしその直前にエスケープ文字‘￥’がある時は置き換えません。

２バイトだったものが１バイトになったり、３バイトだったものが１バイトになることがありますが、その場合はそこから後ろを前の方に詰めることにします。

例えば、”％［ａＡ］＊．ｃ”というパターンは”分月ａＡ日〒．ｃ”と整形されます。

```
/*    make pattern    */

#include <stdio.h>
#include <ctype.h>

#define YES 1
#define NO  0

#define MASK  (0xff80)
#define ANY   '?'
#define NOT   '^'
#define BRA   '['
#define KETT  ']'
#define HIFUN '-'
#define TOP   '%'
#define BOTTOM '$'
#define ESCAPE '¥¥'
#define CLOSE '*'

#define XANY     0xf1
#define XNOT     0xf2
#define XBRA     0xf3
#define XKETT    0xf4
#define XHIFUN   0xf5
#define XTOP     0xf6
#define XBOTTOM  0xf7
#define XCLOSE   0xf8

#define ishex(c)  ( isdigit(c) || (c)>='A' && (c)<='F' || (c)>='a' && (c)<='f')
#define ahtoi(c)  ( ((c)<='9')?(c)-'0':( ((c)<='F')?(c)-'A'+10:(c)-'a'+10 ) )


int makepat( p )
    char *p;
{
    int  c;

    if ( *p == '¥0' )
        return( NO );
    if ( *p == TOP )
        *p++ = XTOP;
    while ( c = *p++ )
        switch ( c ) {
        case ESCAPE:
            if ( ishex( c = *p ) && ishex( p[1] ) ) {
                *(p-1) = ahtoi( c ) * 16 + ahtoi( p[1] );
                packstr( p, p+2 );
            } else if ( c != '¥0' )
                packstr( p-1, p );
            else
                return( NO );
            break;

        case BRA:
            *(p-1) = XBRA;
            if ( *p == NOT )
                *p++ = XNOT;
            while ( ( c = *p++ ) && c != KETT )
                if ( c== ESCAPE && *p == KETT || c==ESCAPE && *p==HIFUN )
                    packstr( p-1, p );
                else if ( *p == HIFUN )
                    if ( c < p[1] ) {
                        *p = c;
                        *(p-1) = XHIFUN;
                        p += 2;
                    } else if ( c == p[1] )
                        packstr( p, p+2 );
                    else {
                        packstr( p-1, p+2 );
                        p--;
                    }
            if ( c == '¥0' )
                p--;
                else
                    *(p-1) = XKETT;
                break;
            case NOT:
                *(p-1) = XNOT;
                break;
            case ANY:
                *(p-1) = XANY;
                break;
            case CLOSE:
                *(p-1) = XCLOSE;
                break;
            case BOTTOM:
                if ( *p == '¥0' ) {
                    *(p-1) = XBOTTOM;
                }
                break;
            default:
                break;
            }
    return( YES );
}



packstr( p, q )
    char *p, *q;
{
    register int  c;

    while ( c = *q++ )
        *p++ = c;
    *p = '¥0';
}
```

次に実際にパターンマッチングする方法を考えましょう。

onematchは1行の入力中にパターンで表される文字列があるかないかを調べる関数です。onematch自身はＸＴＯＰのみを調べ、あとのことはtailmatchにまかせます。tailmatchは入力行を1文字ずつずらしていって、どこかでマッチするところがあるかどうか調べます。

下請け関数pmatchはｓで表される行の中に、ｐで表されるパターン（整形済み）があるかどうかを調べ、ＹＥＳまたはＮＯを返します。ただしｓとｐの第1文字目から比較していきます。

pmatchはmakepatと同じく、case文から成り立っています。'*'の処理は再帰を用いておりpmatch自身を呼び出します。再帰を用いたためにスピードが目に見えて遅くなるということも無かったようなので、この部分の最適化はしていません。

また'['が現れた時は関数brakettelにまかせます。中に条件反転の'^'が入っているかどうかで変数matにＹＥＳまたはＮＯの値を設定しておいて、中にマッチするものがあったかどうかを表すfoundとＸＯＲ（排他的論理和）をとります。

すると全体としてマッチしているかいないか、という値になるのです。

```
/*    match    */

#include <stdio.h>
#include <ctype.h>

#define YES 1
#define NO  0

#define ANY   '?'
#define NOT   '^'
#define BRA   '['
#define KETT  ']'
#define HIFUN '-'
#define TOP   '%'
#define BOTTOM '$'
#define ESCAPE '¥¥'
#define CLOSE '*'

#define XANY     0xfff1
#define XNOT     0xfff2
#define XBRA     0xfff3
#define XKETT    0xfff4
#define XHIFUN   0xfff5
#define XTOP     0xfff6
#define XBOTTOM  0xfff7
#define XCLOSE   0xfff8

#define ishex(c)  ( isdigit(c) || (c)>='A' && (c)<='F' || (c)>='a' && (c)<='f')
#define ahtoi(c)  ( ((c)<='9')?(c)-'0':( ((c)<='F')?(c)-'A'+10:(c)-'a'+10 ) )


int onematch( buf, p )
    char *buf, *p;
{
    if ( *p == XTOP ) {
        return( pmatch( buf, ++p ) ? 1: NO );
    } else
        return( tailmatch( buf, p ) );
}


int tailmatch( s, p )
    char *s, *p;          /*  p: pattern, s: input_line  */
{
    int  n;

    if ( *p == '¥0' )
        return( YES );
    for ( n = 1; *s; n++, s++ )
        if ( pmatch( s, p ) )
            return( YES );
    return( NO );
}

int pmatch( s, p )     /*  string, pattern  */
    char *s, *p;
{
    char *brakette();
    int  c;

    for ( ; c = *p++ ; s++ ) {
        if ( c == XCLOSE ) {
            do {
                if ( pmatch( s, p ) )
                    return( YES );
            } while ( *s++ );
            return( NO );
        } else if ( c == XBOTTOM )
            return( (*s)? NO: YES );
        else if ( *s )
            switch ( c ) {
```

```
            case XBRA:
                if ( ( p = brakette( p, *s ) ) == NULL )
                    return( NO );
                break;
            case XNOT:
                if ( *s == *p++ )
                    return( NO );
                break;
            case XANY:
                break;
            default:
                if ( c != *s )
                    return( NO );
                break;
            }
        else
            return( NO );
    }
    return( YES );
}



char *brakette( p, sc )
    register char *p;
    int  sc;
{
    int  mat, found, c;

    sc &= 0xff;
    mat = ( *p == XNOT )? ( p++, NO ): YES;
    for ( found = NO; ( c = *p ) && c != XKETT; p++ )
        if ( c==XHIFUN &&  p[1] && p[1]!=XKETT && p[2] && p[2]!=XKETT ) {
            if ( sc >= (p[1]&0xff) && sc <= (p[2]&0xff) )
                found = YES;
            p += 2;
        } else if ( sc == c )
            found = YES;
    return( ( mat ^ found )? NULL: (c? ++p: p ) );
}
```

おあずけにしていたメインルーチンを設計します。

第一にパターンを２つ以上置けるようにすることを考えます。そして、すべてのパターンとマッチする、あるいはどれか１つでもマッチする、のどちらと考えるべきかをオプションフラグで指定します。またマッチする条件の反転を指定するフラグおよび行番号を付加するフラグも付けておきます。

| | |
|---|---|
| −a | すべてのパターンとマッチする行 |
| −o | 少なくとも１つのパターンとマッチする行 |
| −v | マッチしない行 |
| −n | 行番号を付加 |

混乱を避けるため、入力ファイルの指定はできないことにします。必ず標準入力から読み込むこととし、コマンドとして使う時はリダイレクトしなければなり

ません。またオプションは必ずパターンの前に置くことにします。これもまた混乱を避けるためです。

パターンがいくつか置かれるか分からないため、パターンの整形の際にいったん他の配列に移すとすると、１つのパターンあたり２００バイト程度として１０パターンまで許すとして２キロバイトのメモリを食います。この無駄を避けるために他の配列を作らない方法をとることは前に述べました。

ファイルを読み込む前にすべてのパターンを整形し、その時NOが帰って来ればエラーとして終了します。

kgrepは'−v'オプションの処理をします。またその下のmatchが'−a'または'− o'の処理をします。'−v'の判定を後にしますから、

```
OS9 : grep  -av  pattern1  pattern2
```

とした場合は、pattern1 と pattern2 の両方ともマッチしない行が出力されることになります。

```
/*    expanded 'grep'    */

/*    Usage: grep <-aonx> (pattern).    */

/*      if using -a or -o option, pattern may laid more than two.

        -a means 'and', include pattern1 AND include pattern2 ...
        -o means 'or', include pattern1 OR include pattern2 ...
        -n means 'line number', print with line number
        -v means 'reverse', if NOT include pattern                   */

/*    SYNTAX:

        '*'      matches any string even NULL string
        '?'      matches any one charactor
        '^a'     matches one charactor exept 'a'
        '%'      matches top point of the line
        '$'      matches bottom point of the line
        '¥'      is escape-sequence,
                     matches followed charactor itself
        '¥1b'    but, if followed two chractors are HEX,
                     matches one charactor it means as ASCII
        '[abc]   matches one charactor 'a' or 'b' or 'c'
        '[^abc]' matches one charactor except 'a', 'b', 'c'
        in [..], 'a-c' maches one charactor within ASCII 'a' to 'c'   */


#include <stdio.h>
#include <ctype.h>

#define YES 1
#define NO  0

char buf[BUFSIZ];
int  andpattern;
int  orpattern;
int  number;
int  reverse;
```

```
main( argc, argv )
    int  argc;
    char **argv;
{
    register char *p;
    int  c, n;

    for ( --argc, ++argv; *(p=*argv) == '-'; argc--, argv++ ) {
        if ( *++p == '¥0' ) {
            usage();
            return;
       }
       while ( c = *p++ )
            switch ( c ) {
            case 'a': andpattern = YES; break;
            case 'o': orpattern = YES;  break;
            case 'n': number = YES;   break;
            case 'v': reverse = YES;   break;
            default:
                usage();
                return;
            }
    }
    if ( argc == 0 || ( andpattern && orpattern )
                   || ( !andpattern && !orpattern && argc >= 2 ) ) {
        usage();
        return;
    }
    for ( n = 0; n < argc ; n++ )
        if ( makepat( argv[n] ) == NO ) {
            fprintf( stderr, "Illegal pattern.¥n" );
            return;
        }
    kgrep( argc, argv );
}


usage()
{
    fprintf( stderr, "Usage: kgrep <-aonv> {pattern} <-aonv>.¥n" );
}



kgrep( n, p )
    int  n;
    char **p;
{
    int  matn;
    int  linenum = 0;

    while ( fgets( buf, BUFSIZ, stdin ) ) {
        linenum++;
        if ( ( matn=match(buf,n,p) ) && !reverse || matn == NO && reverse )
            putline( buf, linenum );
        else if ( matn == NO && reverse )
            putline( buf, linenum );
    }
}



putline( p, n )
    char *p;
    int  n;
{
    if ( number )
        printf( "%05d ", n );
    fputs( buf, stdout );
}



int match( buf, n, p )
    char *buf;
    int  n;
    char **p;
{
    if ( andpattern ) {
        while ( n-- )
            if ( !onematch( buf, *p++ ) )
                return( NO );
        return( YES );
    } else if ( orpattern ) {
        while ( n-- )
            if ( onematch( buf, *p++ ) )
                return( YES );
        return( NO );
    } else
        return( onematch( buf, *p ) );
}
```

# Basic09によるプログラム

# 3—1　パラメータの渡し方

## 3—1—1　I-Code モジュールへのパラメータ

Basic09のプロシジャをＰＡＣＫしたものはI-Codeモジュールというサブルーチンモジュールになっています。I-Codeは一種の中間言語であって６８０９のマシン語ではありませんので、プロセスとして直接起動することはできずに、Runbというランタイムルーチンを必要とします。

Shellコマンドラインで I-Codeモジュールを起動させようとすると、（自動的に）ShellはRunbを起動させ、I-Codeモジュール名そのものはRunbに対するパラメータとみなします。（詳しくは第4章を参照。）このコマンドに対してパラメータを与える場合、Basic09のプログラムを書く時と同様に括弧でくくってやらなければなりません。

例としてファイルの最初の何行かを出力するプログラムheadを挙げます。

```
PROCEDURE head
param file: string[256]
param n: integer
dim line: string[256]
dim path: byte
dim i: integer

    open #path, file: read
    on error goto 10
        for i = 1 to n
            input #path, line
            print line
        next i
    10 on error
    close #path
end
```

これを起動するには、

OS9：head("filename",20)

のようにします。これでファイル"filename"の最初の２０行が表示されるでしょう。ただし、２つの引数のどちらか片方でも省略することはできず、順番を変えることもできません。したがってユーザは、引数の個数、順番およびそれらの型（文字列か数か）を正確に記憶していなければなりません。それがユーザにとっ

てどれほどの負担になるか想像に難くありません。

たとえこれを解決したとしても、もう一つの困難は残ってしまいます。それは引数を与える場合、括弧でくくらなければならないということです。ユーザはコマンドがBasic09のプログラムであるということを覚えていなければならないのです。

## 3―1―2　括弧をはずす

括弧をつけないですませる方法を考えましょう。

括弧は、起動するモジュールがI-Codeモジュールである以上は避けることができません。そこで、発想を少し変えて実行可能モジュールを起動させ、その中でパラメータに括弧をつけさせ、しかるべきI-Codeモジュールを起動させることにすればうまくいきます。

例えば、moduleというコマンドがあったとしましょう。起動する時は、

OS9：module　arg1　arg2　arg3

これによって、moduleには'arg1　arg2　arg3'というパラメータが与えられます。moduleはこれをダブルクォートでくくり、さらに括弧でくくり、こうしてできた文字列の前にI-Codeモジュール名（例えばmodule_main）を置きます。これで、

module_main（"arg1　arg2　arg3"）

というものができます。ここで、この文字列をパラメータとして、Runbを起動(Fork)し、module自らはRunbの終了を待ちます。

```
****************************************
*
*    Basic09's procedure drive program
*
*    give parameter to 'procedure'
*        as one string.
*

****************************************
```

```
       *
       *    Usage:
       *
       *        OS9:module parameter
       *
       *        then 'module' forks 'Runb' giving argument as
       *
       *        'procedure ("parameter")'
       *
       *
       *        'procedure' must have
       *
       *        PARAM parameter:STRING[256]
       *

                                  ifp1
                                  endc

  0000 87CD0067                   mod   Size,Name,Prgrm+Objct,ReEnt+1,Entry,Mem

       ***************************************
       *                                     *
       *    please change these 2 lines.     *
       *                                     *
  000D 6D6F6475     Name      fcs   /module/
  0013 70726F63     PrcName   fcc   /procedure/
       *                                     *
       *                                     *
       ***************************************

  001C 202822                 fcc   / ("/
  001F 00                     fcb   0

  0020 52756EE2     Runb      fcs   /Runb/

D 0000                        org   0
D 0000              Param     rmb   256
D 0100                        rmb   200
D 01C8              Mem       equ   .


  0024              Entry     equ   *
  0024 3410                   pshs  x
  0026 308DFFE9               leax  PrcName,pcr Basic09 procedure name
  002A 31C4                   leay  Param,u    Runb's parameter area
  002C A680         bsh1      lda   ,x+        copy procedure name as Runb's
  002E A7A0                   sta   ,y+
  0030 26FA                   bne   bsh1
  0032 313F                   leay  -1,y
  0034 3510                   puls  x

  0036              bsh2      equ   *
  0036 A680                   lda   ,x+        copy my parameter
  0038 A7A0                   sta   ,y+
  003A 810D                   cmpa  #$0D       end of parameter
  003C 26F8                   bne   bsh2

  003E 8622                   lda   #'"
  0040 A73F                   sta   -1,y
  0042 CC290D                 ldd   #')*256+$0D
  0045 EDA1                   std   ,y++

  0047 3440                   pshs  u
  0049 1F20                   tfr   y,d
  004B A3E1                   subd  ,s++
  004D 1F02                   tfr   d,y
       *
       *    U reg. means parameter address
       *    Y reg. means parameter size
       *
  004F 308DFFCD               leax  Runb,pcr   name of Runb
  0053 8611                   lda   #Prgrm+Objct
  0055 5F                     clrb
  0056 103F03                 os9   F$Fork
  0059 2506                   bcs   exit
  005B 103F04                 os9   F$Wait
```

```
00087   005E 2501                bcs    exit
00088   0060 5F                  clrb
00089   0061 103F06     exit     os9    F$Exit
00090
00091
00092   0064 080A9E              emod
00093   0067            Size     equ    *
00094                            end
```

これによってI-Codeモジュールであるmodule_mainには1個の文字列が渡されます。したがってmodule_mainの冒頭では次のように宣言しておきます。

param　argument：　string［256］

安全のためにサイズを２５６としてあります。
パラメータが1個の文字列として渡されることからわかるように、パラメータの解釈はプログラムの自由です。次節でそれについて考えることにします。

さて、module.aのソースを見ればわかるように、実際に使用する時は行３５－３６を必要に応じて書き換えてアセンブルして下さい。

一般にBasic09のプログラムは複数のプロシジャから構成され、ＰＡＣＫすると複数のモジュールが生成されます。module.aを書き換えてアセンブルしたオブジェクトモジュールと、ＰＡＣＫしたI-Codeモジュールをすべてマージし、ＣＭＤＳにコピーすればコマンドの出来上がりです。(この時アトリビュートを実行可能にしておくのを忘れないようにして下さい。)

## 3－1－3　パラメータの解釈

パラメータが唯一個の文字列として渡されるため、その中から各単語を取り出す作業が必要です。たいていの場合、スペースがセパレータとして使用されますので、本書でもそれに倣います。

分解の仕方を考えてみましょう。

まず第一に、C言語の方法です。Cでは引数は直接渡されず、argcおよびargvが渡されます。argcは引数の個数を表し、argvは各引数へのポインタから成る配列です。Basic09にはポインタの概念が無いので、もとの1個の文字列をキャラクタの配列と考え、そのインデックスで表したらよいように思えます。しかし、Cとは違ってBasic09では可変長の文字列が非常に扱いにくく（Pascalの仕様をもとにしているため)、そこから各引数を復元するために手間がかかってしまいます。

本書では次に述べる第二の方法をとることにしました。すなわちgetargというPROCEDUREを通じて各引数を取り出します。

getargは与えられた文字列の最初の単語を取り出すと同時に、取り出した残りを新たな文字列とします。例えば、

argument＝" arg 1　arg 2　arg 3 "

にgetargを繰り返し適用します。

| 取り出す前 | 単語 | 取り出した後 |
|---|---|---|
| “arg1　arg2　arg3 | "arg1" | "arg2" arg3" |
| "arg2　arg3" | "arg2" | "arg3" |
| "arg3" | "arg3" | "" |
| "" | "" | "" |

単語の頭のスペースはスキップされます。

もし引数がすべてファイル名であるなら、（一つのファイルについてのプロシ

ジャをonefileとすれば）次のようなプログラムができるでしょう。

```
PROCEDURE module_main
param argument: string[256]
dim arg: string[256]

    loop
        run getarg( argument, arg )
        exitif arg = "" then
        endexit
        run onefile( arg )
    endloop
end
```

またもしオプションフラグを含むのであればloop-endloopの前にもう一つloopを置き、その中でフラグの設定等を行えばよいでしょう。その際に気を付けることは、getargはもとの文字列を書き変えてしまうために、もとのままをどこかに残しておかなければならないということです。もっと良い方法は、オプションの解釈を別のプロシジャで行ってしまうことです。その時、

argument＋""

として値渡しにしてしまえばargumentは書き変えられません。

```
00001        ****************************************
00002        *
00003        *    RUN getarg( line, arg )
00004        *        param line, arg: string[???]
00005        *
00006
00007                                ifp1
00009                                endc
00010
00011   0000 87CD0099                mod   Size,Name,Sbrtn+Objct,ReEnt+1,Entry,Mem
00012   000D 67657461   Name         fcs   /getarg/
0Q013   0020            Mem          equ   32
00014
00015   0020            BLANK        equ   $20
00016   00FF            EOS          equ   $FF
00017
00018
00019 D 0000                         org   0
00020 D 0000            RetAdr       rmb   2
00021 D 0002            NPar         rmb   2
00022 D 0004            LinAdr       rmb   2
00023 D 0006            LinSiz       rmb   2
00024 D 0008            ArgAdr       rmb   2
00025 D 000A            ArgSiz       rmb   2
00026
00027
00028   0013            Entry        equ   *
00029   0013 EC62                    ldd   NPar,s
00030   0015 10830002                cmpd  #2
00031   0019 266E                    bne   parerr     parameter error
00032
00033   001B ECF804                  ldd   [LinAdr,s]
```

```
001E E3F806                   addd  [LinSiz,s] end address of 'arg'
0021 1F01                     tfr   d,x
0023 ECF808                   ldd   [ArgAdr,s]
0026 E3F80A                   addd  [ArgSiz,s] end address of 'line'
0029 3416                     pshs  d,x

002B AE68                     ldx   LinAdr+4,s
002D 8D5E                     bsr   skip       return D: size
002F ACE4                     cmpx  ,s
0031 250C                     bcs   getarg1

0033 3264                     leas  4,s
0035 86FF                     lda   #EOS
0037 A7F804                   sta   [LinAdr,s]
003A A7F808                   sta   [ArgAdr,s]
003D 5F                       clrb
003E 39                       rts

003F                getarg1   equu  *
003F 10AE6C                   ldy   ArgAdr+4,s

0042                getarg2   equ   *
0042 ACE4                     cmpx  ,s
0044 2413                     bcc   getarg3
0046 10AC62                   cmpy  2,s
0049 2438                     bcc   tooshort   'arg' is too short to copy 'li
004B A680                     lda   ,x+
004D 81FF                     cmpa  #EOS
004F 2708                     beq   getarg3
0051 8120                     cmpa  #BLANK
0053 2706                     beq   getarg4
0055 A7A0                     sta   ,y+
0057 20E9                     bra   getarg2

0059 301F           getarg3   leax  -1,x
005B 10AC22         getarg4   cmpy  2,y
005E 2404                     bcc   getarg5
0060 86FF                     lda   #EOS
0062 A7A4                     sta   ,y

0064 ACF808         getarg5   cmpx  [LinAdr+4,s]
0067 2716                     beq   getarg8
0069 10AEF808                 ldy   [LinAdr+4,s]
006D ACE4           getarg6   cmpx  ,s
006F 240A                     bcc   getarg7
0071 A680                     lda   ,x+
0073 81FF                     cmpa  #EOS
0075 2704                     beq   getarg7
0077 A7A0                     sta   ,y+
0079 20F2                     bra   getarg6
007B 86FF           getarg7   lda   #EOS
007D A7A4                     sta   ,y

007F                getarg8   equ   *
007F 3264                     leas  4,s
0081 5F                       clrb
0082 39                       rts


0083                tooshort  equ   *
0083 3264                     leas  4,s
0085 53                       comb
0086 C634                     ldb   #52        out of range
0088 39                       rts


0089                parerr    equ   *
0089 53                       comb
008A C638                     ldb   #56
008C 39                       rts


008D                skip      equ   *
008D A680                     lda   ,x+
008F 8120                     cmpa  #BLANK
0091 27FA                     beq   skip
0093 301F                     leax  -1,x
```

```
0095 39                       rts


0096 44D290                   emod
0099               Size       equ    *
                              end
```

さて、Basio09でコマンド記述した場合、たいていはgetarg を使うと思われます。筆者の場合はgetargだけ独立して（Startupで）loadし、メモリモジュールに常駐させています。このようにしておけば、コマンドを作るごとにgetargをマージする必要もなく、かなりの手間が省けています。

## 3－1－4　プロシジャ名の衝突

Basic09はプロシジャの集まりとして記述されることは述べました。プログラムのモジュール化という点では大変に結構なことですが、反面どんな小さなプロシジャでもモジュールとして登録されていますので、多くのコマンドを作っていくうちに、以前作ったものと同じ名前のプロシジャができてしまう可能性があります。また基本的な関数がすべて備わっているわけではないので、各コマンドでまったく同じプロシジャを使おうとするかもしれません。

もし2つ以上のコマンドが同じ名前のプロシジャを持っていたとすると、少しばかり厄介なことになります。なぜなら、パッケージ化したファイルをロードする時、もしその中の一つと同じ名前のモジュールがすでにモジュールディレクトリ上にあると、その時点でロードするのを止めてしまうのです。したがってそれ以後のモジュールはロードされないため、いざ実行しようとすると、プロシジャが無いというエラーになってしまいます。

これを避ける手段は2つしかありません。第一の方法はそういうコマンドを始めから全部メモリ上に常駐させておくことです。また第二の方法は、同じ名前のプロシジャを作らないことです。

前者はランタイムパッケージを置くということですから、全コマンドにおける共有財産と考えることができます。前節での getarg はまさにそういう考えだった訳です。このようなパッケージをたくさんメモリに常駐させておけば、トータルでのメモリ効率も向上するでしょう。これらは Runb の拡張部分であると考えて

もよいかもしれません。しかしいったん常駐させておくことに決めてしまうとそれ以後は必ず常駐させておかなければならないことになります、ちょっとメモリが少なくなったからunlinkしようというわけにはいかないのです。

後者は、メモリに常駐させるためには相当の無駄が生じますが、そうでなければかえってメモリの使用量が減ります。またすでに同じ内容のモジュールがメモリ上にあったとしても、名前が違っていれば支障はありません。

安全のためには後者の方がよいと思われるので、本書では同じ内容のモジュールであっても、各コマンドに異なった名前を与えることにしました。と言ってもまったく似ても似つかない名前ではかえって間違いが起こるでしょう。また、ついうっかり同じ名前を使ってしまうことがあるかもしれません。

これを解決するため、プロシジャの名前の付け方に制限をつけます。すなわち、コマンドの名前をプロシジャの名前の前に必ず付けることにするのです。例えば、wildコマンド中のmatchプロシジャの名前はwild_matchとするのです。まったく同じ内容のものでも、findコマンドで使用する場合はfind_matchという別の名前を付けます。

これらは、各コマンドごとにＰＡＣＫしてマージし、ＣＭＤＳに格納しておきます。

なおこのどちらの方法を選ぶかはシステムの状態によっても違います。もし、前者の方法をとってメモリに常駐させるのであれば、本書のプロシジャ名の一部を変更して下さい。

## 3―2　プログラム例

### 3―2―1　err

Basic09をコマンド記述言語に仕立てあげるための最初の例としてとして、エラーメッセージを出力するコマンドを作ってみます。といっても、エラー番号を指定すると該当するエラーメッセージを返すだけのものです。

使い方は、

```
OS9：err　214　216
```

と、エラー番号をパラメータとして並べます。

プログラムは/d0/sys/errmsgをサーチしているだけの単純なものですが、なかなか重宝するコマンドです。

レベル２にはprinterrコマンドを載せておらず、エラーメッセージを見たい時にはerrmsgファイルをgrepで覗くなどという方法を取らざるを得ません。また、レベル１にしても、printerrをunlinkすることができないために、一度printerrを実行してしまうと電源を切るまで解除できないという煩わしさがあります。持にフロッピベースではエラーの度にディスクをアクセスしに行きますから時間もかかり、うるさくもあります。

さてプログラムはまず２５６個のＢＯＯＬＥＡＮ型の配列を用意し、パラメータで指定された位置をＴＲＵＥに、それ以外はＦＡＬＳＥにセットします。つまりこの配列がエラー０から２５５に対応しており、ＴＲＵＥのエラーだけ表示してもよいということを表しているのです。ただし、エラー０は存在しませんのでＦＡＬＳＥにしておきます。(どうせ存在しないのですからＴＲＵＥになっていても別に不都合は生じませんが。)

次に/d0/sys/errmsgをオープンし１行ずつ読み込んでいきます。errmsgは次のような書式になっています。

```
エラー番号　――　エラーメッセージ
```

この１行をＳＴＲＩＮＧ型のlineに代入し、ＶＡＬをとればエラー番号がＩＮＴＥＧＥＲとして得られます。この番号をインデックスとして先の配列をみてＴＲＵＥだったらlineをそのまま出力します。

以上からわかるようにerrmsgに存在しないエラーは出力できません。

```
PROCEDURE err_main
 (*    err  --  print errmsg file     *)
 (*    usage: err (error_number).     *)
base 0
param argument: string[256]
dim line: string[256]
dim init, table(256): boolean
dim i: integer; path: byte

    argument = trim$( argument )
    if argument = "" then
        init = TRUE
    else
        init = FALSE
    endif
    for i = 0 to 255
        table(i) = init
    next i
    loop
        run getarg( argument, line )
        exitif line = "" then
        endexit
        on error goto 999
            table( val( line ) ) = TRUE
        on error
    endloop
    table(0) = FALSE

    open #path, "/d0/sys/errmsg": read
    while not eof(#path) do
        input #path, line
        on error goto 10
            if table( val( line ) ) then
                print line
            endif
        10 on error
    endwhile
    close #path
    end
999
    print #2, "usage: err ( err_num )."
end
```

## 3—2—2　mh

mhはパッケージ化されたモジュールファイル中の、各モジュールの名前およびヘッダ情報を出力するプログラムです。

identでも同じようなことはできますが、その出力を他の用途に使うことはできません。mhはオプション無しの時はモジュール名のみを出力し、オプションと

して'－e'をつけた時にヘッダ情報を出力することにします。このように仕様を決めますと次のような使い方ができるようになります。(これは実際に筆者のシステムで頻繁に行っている作業です。)

レベル２ではコマンドをパッケージ化しておくとメモリの節約になるということは前にも触れました。では効率的にパッケージ化するにはどうしたらよいでしょうか。レベル２のメモリ割り当ては４ＫＢ単位で行われるため、１つのパッケージのサイズを４ＫＢの倍数ぎりぎりに取るのがよいでしょう。ただし、何でも詰めこんでしまえばよいかというと、そうではありません。ある程度のかけひきのようなものが必要です。結局４ＫＢまたは８ＫＢのものを複数個作るのが最も効率的だと思われます。

問題はいったん作った後で、変更の必要が生じたときにどうするかです。書き換えるべきプログラムをどこに入れておいたかを見るために（そんなことをいちいち覚えていないことが多いですから)、

```
OS9：mh　－ex　attr. etc　data. etc　exbin. etc
```

または次のようにします。

```
OS9：wild　/d0/cmds/＊. etc　!　place　mh　－ex　＊　!　shell　－p
```

```
'attr.etc'
module name        size  type    lang     at/rv
-----------------  ----  ------  ------   -- --
Attr               0286  Prgrm   Objct    80 01
Build              0054  Prgrm   Objct    80 01
Del                00A5  Prgrm   Objct    80 01
Dir                0300  Prgrm   Objct    80 01
Display            0084  Prgrm   Objct    80 01
Echo               0022  Prgrm   Objct    80 01
Link               002C  Prgrm   Objct    80 01
Makdir             0022  Prgrm   Objct    80 01
MDir               026A  Prgrm   Objct    80 01
Merge              0068  Prgrm   Objct    80 01
Mfree              01EB  Prgrm   Objct    80 01
Procs              0280  Prgrm   Objct    80 01
Rename             011D  Prgrm   Objct    80 01
Unlink             002D  Prgrm   Objct    80 01

'date.etc'
module name        size  type    lang     at/rv
-----------------  ----  ------  ------   -- --
Date               00FD  Prgrm   Objct    80 01
Deldir             027C  Prgrm   Objct    80 01
Free               02D1  Prgrm   Objct    80 01
pwd                0130  Prgrm   Objct    80 01
pxd                0130  Prgrm   Objct    80 01
Tee                007D  Prgrm   Objct    80 01
```

```
Tmode            0207  Prgrm  Objct   80 01
lc               00CD  Prgrm  Objct   80 01
lin              00C9  Prgrm  Objct   80 01
lower            008F  Prgrm  Objct   80 01
upper            008F  Prgrm  Objct   80 01
rev              0099  Prgrm  Objct   80 01
hex.             00B5  Prgrm  Objct   80 01

'exbin.etc'
module name      size  type   lang    at/rv
---------------- ----  ------ ------  -- --
Exbin            01BA  Prgrm  Objct   80 01
Binex            0278  Prgrm  Objct   80 01
Dump             0186  Prgrm  Objct   80 01
Verify           0172  Prgrm  Objct   80 01
Color.           012D  Prgrm  Objct   80 01
Key.             023C  Prgrm  Objct   80 01
Wset             0193  Prgrm  Objct   80 01
fdump            0343  Prgrm  Objct   80 01
Save             0067  Prgrm  Objct   80 01

'kset.etc'
module name      size  type   lang    at/rv
---------------- ----  ------ ------  -- --
Kset             00DD  Prgrm  Objct   80 01
wc               00F6  Prgrm  Objct   80 01
print            0367  Prgrm  Objct   80 01
fcmp             02AA  Prgrm  Objct   80 01
mdump            027D  Prgrm  Objct   80 01
Xmode            0391  Prgrm  Objct   80 01
Rename           011D  Prgrm  Objct   80 01

'pmode.etc'
module name      size  type   lang    at/rv
---------------- ----  ------ ------  -- --
Pmode            0355  Prgrm  Objct   80 01
Sleep            004D  Prgrm  Objct   80 01
D                048B  Prgrm  Objct   80 01
Ident            06E7  Prgrm  Objct   80 01
hex              008B  Sbrtn  Objct   80 01

'graphics.etc'
module name      size  type   lang    at/rv
---------------- ----  ------ ------  -- --
Cls              00BC  Sbrtn  Objct   80 01
Color            00CE  Sbrtn  Objct   80 01
Connect          0134  Sbrtn  Objct   80 01
Key              00E4  Sbrtn  Objct   80 01
KPrint           013F  Sbrtn  Objct   80 01
KSymbol          0184  Sbrtn  Objct   80 01
Line             01F3  Sbrtn  Objct   80 01
Locate           00FD  Sbrtn  Objct   80 01
Paint            016B  Sbrtn  Objct   80 01
Pset             0109  Sbrtn  Objct   80 01
Symbol           015A  Sbrtn  Objct   80 01
```

例えばlinを取り替えるのであれば、attr. etcをapartで分解し、修正の後でmergeしてattr. etcそのものを取り換えます。実際の修正の手順はapartの項を参照して下さい。

さて、オプションとして'－x'と'－e'の2つを設けます。

'－e'はdirコマンドの'e'オプションと同じで、より詳しい情報を出力するべきことを表します。'－x'はファイルが実行ディレクトリにあることを表します。

オプションはグローバル変数として扱いたいために、extern というＧＬＯＶＡＬ型の変数の中にＢＯＯＬＥＡＮ型の execute と expand として宣言します。グローバルと言っても本当にグローバルなのではなく、疑似的なものであるにすぎません。したがって、引数として最下位のＰＲＯＣＥＤＵＲＥにまで渡していかなければなりません。

extern にはこの他に tps と lgs という文字列配列を含みます。この２つにはそれぞれ後で述べる、「タイプ／言語」名を表す文字列をあらかじめセットしておきます。これはプログラム中ではずっと定数値として扱われますので「変数」とは思わないようにして下さい。Basic09ではグローバル変数が扱えない他に、変数の初期化ができないということがあり、初期化をするためにはＤＡＴＡ文を使うしかありません。出力する時になってＤＡＴＡ文を読んでいくのでは手間がかかりすぎますから、配列に入れておいて後は１回で読み出すということにしましょう。

もう１つの extern の要素として head という構造体がありますが、これは実はローカル変数として使用されています。これはメモリ節約のための苦肉の策なのです。すべての変数をローカルに宣言すると、ＰＲＯＣＥＤＵＲＥ呼び出しの多重度が増すごとにスタックの使用量が増えて、最後にはＥＲＲＯＲ　＃２０７(メモリフル）で止まってしまうことがありますので、このような事態を未然に防ぐためにローカルな変数をグローバルに宣言しておくという方法をとっているのです。もっとも mh 程度では止まりはしないでしょうが、よりメモリを浪費するであろうことは変わりありません。

さて各モジュールのタイプ／言語を表示する時、単に１６進で出すよりはそれを表す文字列で出した方がよいでしょう。例えば、「＄１１」と表示されるよりは「Prgrm＋Objct」と出たほうが分かりやすいです。ここではこれらの文字としてアセンブラの時に定義してあった次のものを使用します。

| | タイプ | 言語 |
|---|---|---|
| 0 1 | prgrm | Objct |
| 0 2 | Sbrtn | ICode |
| 0 3 | Multi | PCode |
| 0 4 | Data | CCode |
| 0 5 | | CblCode |
| 0 6 | | FrtnCode |
| 0 8 | | |
| 0 9 | | |
| 0 A | | |
| 0 B | | |
| 0 C | Systm | |
| 0 D | Flmgr | |
| 0 E | Drivr | |
| 0 F | Devic | |

未定義のものが現われた時は１６進で表示することにします。

さて、１つの入力ファイルを処理するのが mh_onefile です。ファイルをオープンするために mh_openfile を呼び出しますが、そこでは'－x'オプションがあったかどうかによって、ＲＥＡＤあるいはＲＥＡＤ＋ＥＸＥＣでＯＰＥＮします。これはＣにおける次のものとほぼ同様の働きをします。

(exec ? fopen(ファイル名, "rx")：fopen(ファイル名,"r"))

処理すべきファイルは、extern. execute がＴＲＵＥならば実行ディレクトリから捜し、ＦＡＬＳＥならばワーキングディレクトリから捜します。ただし、ディレクトリファイルだとすると普通にオープンしたのではエラーになってしまいます。その場合にもエラーストップせずに正常に処理を続けるには Basic09を使う限り ＯＮ ＥＲＲＯＲ ＧＯＴＯより他に方法がありません。

```
PROCEDURE mh_main
base 0
param argument: string[ 256 ]
type header = modname: string[32]; siz, tp, lg, at, rev, hp: integer
type GLOVAL = head: header; tps(16),lgs(16):string[8]; execute,expand:boolean
dim extern: GLOVAL
dim arg: string[ 256 ]

    shell "link mh_main mh_option mh_onefile mh_onemodule"
    run mh_option( extern, argument+"")
    loop
        run getarg( argument, arg )
        exitif arg = "" then
        endexit
        if asc( arg ) <> $2D then
            run mh_onefile( extern, arg )
        endif
    endloop
    shell "unlink mh_main mh_option mh_onefile mh_onemodule"
end


PROCEDURE mh_option
base 0
type header = modname: string[32]; siz, tp, lg, at, rev, hp: integer
type GLOVAL = head: header; tps(16),lgs(16):string[8]; execute,expand:boolean
param extern: GLOVAL; argument: string[256]
dim arg(256): byte; i: integer

    extern.execute = FALSE
    extern.expand = FALSE
    loop
        run getarg( argument, arg )
        exitif arg(0) = $FF then
        endexit
        if arg(0) = $2D then
            for i = 1 to 255
                exitif arg(i) = $FF then
                endexit
                if arg(i) = $45 or arg(i) = $65 then ¥(* -e *)
                    extern.expand = TRUE
                else if arg(i) = $58 or arg(i) = $78 then ¥(* -x *)
                    extern.execute = TRUE
                endif ¥ endif
            next i
        endif
    endloop

    for i = 0 to 15
        read extern.tps(i)
    next i
    for i = 0 to 15
        read extern.lgs(i)
    next i
    end

    data "00"
    data "Prgrm", "Sbrtn", "Multi", "Data"
    data "50", "60", "70", "80", "90", "a0", "b0"
    data "Systm", "Flmgr", "Drivr", "Devic"
    data "00"
    data "Objct", "ICode", "PCode", "CCode", "CblCode", "FrtnCode"
    data "07", "08", "09", "0a", "0b", "0c", "0d", "0e", "0f"
end


PROCEDURE mh_onefile
base 0
type header = modname: string[32]; siz, tp, lg, at, rev, hp: integer
type GLOVAL = head: header; tps(16),lgs(16):string[8]; execute,expand:boolean
param extern: GLOVAL; name: string[256]
```

```
dim position: real
dim nam: integer; path: byte; mode: integer

    run mh_openfile( extern, path, name, mode )
    if mode = -1 then    (* can't open *)
        print #2, "can't open " + chr$($22) + name + chr$($22)
        end
    else if mode = 2 then (* directory *)
        print using "s17<", name;
        print "<d>"
    else
        position = 0.
        if extern.expand then
            print
            print "'" + name + "'"
            print "module name      size  type    lang     at/rv"
            print "---------------- ----  ------  ------    -- --"
        endif
        loop
            run mh_onemodule( extern, path, position, mode )
            exitif mode <> 0 then    ¥(* end of file or error *)
            endexit
            if extern.expand then
                print using "s17<",          extern.head.modname;
                print using "h4^^",          extern.head.siz;
                print using "x2,s8<",        extern.tps(extern.head.tp);
                print using "s8<",           extern.lgs(extern.head.lg);
                print using "x1,h2^,x1,h2^^", extern.head.at, extern.head.rev
            else
                print extern.head.modname
            endif
        endloop
        IF MODE <> 0 AND MODE <> -1 then
            PRINT #2, "error #"; mode
        ENDIF
    endif ¥ endif
    close #path
end


PROCEDURE mh_openfile
base 0
type header = modname: string[32]; siz, tp, lg, at, rev, hp: integer
type GLOVAL = head: header; tps(16),lgs(16):string[8]; execute,expand:boolean
param extern: GLOVAL; path: byte; name: string[256]; mode: integer

    on error goto 10
    if extern.execute then
        open #path, name: read+exec
    else
        open #path, name: read
    endif
    mode = 1
    end

 10 on error goto 20
    if extern.execute then
        open #path, name: read+dir+exec
    else
        open #path, name: read+dir
    endif
    mode = 2
    end

20 mode = -1
end


PROCEDURE mh_onemodule
base 0
type header = modname(32): byte; siz, tp, lg, at, rev, hp: integer
type GLOVAL = head: header; tps(16),lgs(16):string[8]; execute,expand:boolean
param extern: GLOVAL; path: byte; position: real; mode: integer
type moduleheader = msync, msiz, mnam: integer; mtplg, matrv, mhp: byte
dim mhead: moduleheader
dim n: integer; c: byte

    extern.head.modname(0) = $FF
    on error goto 10    ¥(* if error, maybe EOF *)
    get #path, mhead
    on error goto 20
    if mhead.msync = $87CD then
        extern.head.siz = mhead.msiz
        extern.head.tp = fix( mhead.mtplg / 16 )
        extern.head.lg = fix( land( mhead.mtplg, $0F ) )
        extern.head.at = land( mhead.matrv, $F0 )
        extern.head.rev = land( mhead.matrv, $0F )
        extern.head.hp = mhead.mhp
        seek #path, position + mhead.mnam
        n = 0
        repeat
            get #path, c
            extern.head.modname(n) = land( c, $7F ) ¥ n = n + 1
        until land( c, $80 ) <> 0
        extern.head.modname(n) = $FF
        position = position + mhead.msiz
        seek #path, position
        mode = 0    ¥(* no error *)
    else
        mode = 1   ¥(* not module error *)
    endif
    end

10
    mode = err    ¥(* error *)
    if mode = 211 then
        mode = -1
    endif
    end

20
    mode = err
end
```

## 3－2－3 para

paraは2つのファイルを横に並べて出力するコマンドです。

ファイルの数は2つに限定してありますが、片方を標準入力することもできます。その場合は第一番目のファイルが標準入力であると見なします。

オプション'－n'を置きますと行番号を付けて出力します。

横は合わせて80桁と仮定しており、行番号がない時は39桁と39桁、行番号がある時は36桁と36桁で表示します(行番号が6桁)。ただし、どちらの場合も2つのファイルの出力の間に1個の空白を挟みます。

実際には次のように実行されます。

```
OS9 : para para_main   para_main  !  head  -10
```

```
PROCEDURE para_main                      PROCEDURE para_main
(*                                       (*
      *)                                       *)
(*    usage: para <-n> <file1> file2 <-  (*    usage: para <-n> <file1> file2 <-
n>.   *)                                 n>.   *)
(*                                       (*
      *)                                       *)
base 0                                   base 0
param argument: string[ 256 ]            param argument: string[ 256 ]
dim name(2): string[128]                 dim name(2): string[128]
```

```
OS9: para  para_main  -n  <  para_main  !  head  -10
```

```
0001 PROCEDURE para_main                      PROCEDURE para_main
0002 (*                                       (*
0003             *)                                     *)
0004 (*    usage: para <-n> <file1> file2  (*    usage: para <-n> <file1> file2
0005  <-n>.      *)                         <-n>.      *)
0006 (*                                       (*
0007             *)                                     *)
0008 base 0                                   base 0
0009 param argument: string[ 256 ]            param argument: string[ 256 ]
0010 dim name(2): string[128]                 dim name(2): string[128]
```

さて para_main では para_option を呼び出してオプションの設定をします。この時コマンドラインそのものを値渡しにしていますので argument は変化していません。なお、オプションにエラーがあった場合には opterr がＴＲＵＥになって帰ってきます。

この後ファイル名を２つ取り出します。２つ見つけた時点でパラメータの走査を止めますので、３つ以上あった場合は最初の２つのみが有効になります。逆に１つしかなかった場合は第一のファイルが標準入力であると見なし、その印として name(0) はＮＵＬＬにしておきます。もしファイル名が１つも置かれていなかった場合はエラーとします。

こうして得られた２つのファイルをオープンしますが、もしＯＰＥＮでエラーだった場合に、どちらがエラーだったかを書き出すためにエラートラップ（ＯＮ　ＥＲＲＯＲ　ＧＯＴＯ文）を置いてあります。必ずしも必要というわけではありませんので取り除いてしまっても構いません。この時、第一のファイルが標準入力と指定されているのであればオープンせずに番号０を使用します。

実際にファイルからの入力および出力をするのが para_para です。まず行番号の指定があったかどうかによって出力の横幅を３６ないし３９に設定します。両

方がＥＯＦに達するまでループしますが、その中はまた3つの部分に分かれています。ファイル1がＥＯＦの場合、ファイル2がＥＯＦの場合、両方ともＥＯＦでない場合です。

また、ＩＮＰＵＴやＲＥＡＤではうまくいかないため、1行を読み込むためにpara_getsを使っています。これは最大数を指定した1行読み込みを行いますので、Cにおけるfgetsに相当するものです。

```
PROCEDURE para_main
 (*                                                  *)
 (*    usage: para <-n> <file1> file2 <-n>.          *)
 (*                                                  *)
base 0
param argument: string[ 256 ]
dim name(2): string[128]
dim path1, path2: byte
dim lineno, errnum: integer
dim number, opterr: boolean
dim i: integer

    shell "link para_option para_para para_gets"
    run para_option( argument+"", number, opterr )
    if opterr then
        print #2, "usage: para <-n> <file1> file2 <-n>."
        end
    endif
    name(0) = "" ¥ name(1) = ""
    for i = 0 to 1
        repeat
            run geterg( argument, name(i) )
        until name(i) = "" or left$( name(i), 1 ) <> "-"
        exitif name(i) = "" then
            name(i) = ""
        endexit
    next i
    if name(0) = "" and name(1) = "" then
        print #2, "usage: para <-n> file1 file2 <-n>."
        end
    else if name(1) = "" then
        name(1) = name(0)
        name(0) = ""
    endif ¥ endif

    on error goto 999
        if name(0) <> "" then
            open #path1, name(0): read
        else
            path1 = 0
        endif
        open #path2, name(1): read
    on error
    run para_para( path1, path2, number )
    if name(0) <> "" then
        close #path1
    endif
    close #path2
    shell "unlink para_option para_para para_gets"
end

999 rem
    print #2, "can't open."; chr$($22); name(i); chr$($22)
    shell "unlink para_option para_para para_gets"
    end
```

```
PROCEDURE para_option
base 0
param argument: string[256]
param number, opterr: boolean
dim option: string[256]

    number = FALSE
    opterr = FALSE
    loop
        run getarg( argument, option )
        exitif option = "" then
        endexit
        if asc( option ) = $2d then
            if option = "-n" or option = "-N" then
                number = TRUE
            else
                opterr = TRUE
                end
            endif
        endif
    endloop
end



PROCEDURE para_para
base 0
param path1, path2: byte
param number: boolean
dim line1, line2: string[100]
dim lineno, width: integer
dim flag: boolean

    if number then
        width = 36
    else
        width = 39
    endif
    lineno = 0
    loop
        run para_gets( line1, width, path1, flag )
        exitif flag then
            loop
                run para_gets( line2, width, path2, flag )
                exitif flag then
                endexit
                lineno = lineno + 1
                if number then
                    print using "i5^,s38", lineno, " ";
                else
                    print using "s40", " ";
                endif
                print line2
            endloop
        endexit
        run para_gets( line2, width, path2, flag )
        exitif flag then
            lineno = lineno + 1
            if number then
                print using "i5^", lineno;
                print " ";
            endif
            print line1
            loop
                run para_gets( line1, width, path1, flag )
                exitif flag then
                endexit
                if number then
                    print using "i5^", lineno;
                    print " ";
                endif
                print line1
            endloop
        endexit
        lineno = lineno + 1
        if number then
```

```
            print using "I5^,x1,s37", lineno, line1;
        else
            print using "s40", line1;
        endif
        print line2
    endloop
end



PROCEDURE para_gets
base 0
param line( 256 ): byte
param max: integer
param path: byte
param flag: boolean
dim i: integer
dim s: string[ 1 ]

    flag = FALSE
    on error goto 999
    for i = 0 to max - 1
        get #path, s
        exitif s = chr$( 13 ) then
            line( i ) = $FF
        endexit
        line( i ) = asc( s )
    next i
    line( max ) = $FF
    end

999 errnum = err
    if errnum = 211 then
        line( i ) = $FF
        flag = TRUE
    else
        error errnum
    endif
end
```

## 3－2－4　time

time は画面右上にリアルタイムの時計を表示させるという、F－BASICのCLOCKコマンドに相当するものですが、FM－11専用で、もしFM－7、8で実行するとシステムがクラッシュします。

コマンド名はF－BASICに合わせてclockとしたかったのですが、clockはすでにモジュールとして存在しますのでtimeという名前にしました。

```
OS9：time　on
```

で、クロックをhh：mm：ssの形で表示し、

```
OS9：time　off
```

でクロックの表示を止めます。

FM－11のOS－9では、サブシステムのクロックをまったく使用していません。つまりサブのタイマはまったく遊んでいるのです。このコマンドは遊んでいるサブのタイマを使いますから、メインへはほとんど負担がかかりません。画面の表示スピードに若干の影響が出るか出ないかといった程度です。

さてプログラム自体は簡単で、あるデータを標準出力へ書き出しているにすぎません。あるデータとはサブコマンドのことですが、グラフィックが単なるデータとして読み書きできるのとまったく同じにあらゆるサブコマンドがデータとして処理できることを利用しています。これらのデータはVTERMドライバが解釈して実際にサブシステムへ送り込んでいるのです。

FM－11のOS－9はサブのタイマを使用していないと言いましたが、逆に言うとサブのタイマは時間が設定されていないということですから、time　onの際にはまずサブのタイマを設定してやらなければならないのです。そのためには次のようなset　timeサブコマンドを使います。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| ＄００ | ＄００ | ＄３Ｄ | ＲＳ | ＴＣ | ｈｈ | ｍｍ | ｓｓ | ｘｘ |

ＲＳ：レジスタ選択フラグ
ＴＣ：制御レジスタ
hh：mm：ss：xx　時：分：秒：20ms

ここで注意してもらいたいのは、４バイト目です。これは制御レジスタで、メインに対する割り込みのオンオフを指定するものです。ＦＭ－１１ではサブのタイマ割り込みを使用しませんので、このバイトは＄００にしておかなければなりません。したがってＦＭ－７、８でこの通りにするとサブからの割込みが来なくなって、カーネルはディスパッチができなくなり、システムはストップしてしまいます。

現在時刻はBasic09ではdate$で得ることができますが、date$は文字列ですからvalで数値に変換しなければなりません。この、現在時刻を上のset　timeサブコマンドで設定するのがプロシジャtime_settimeです。

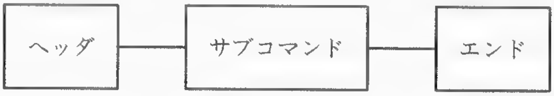
メインルーチンはtime_settimeの後display　clockサブコマンドを実行しますが、これは次のようなものです。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| ＄００ | ＄００ | ＄４０ | Ｆ | ＣＬ | Ｘ | Ｙ |

Ｆ：表示フラグ
ＣＬ：表示カラー
Ｘ：Ｘ座標（０－７９）
Ｙ：Ｙ座標（０－２４）

表示フラグFが0でない場合に表示となります。このX座標とY座標をどこにするかは問題ですが、ここでは画面右上のなるべく端に固定することにしました。そうでないと画面がスクロールされる時にクロック表示も一緒にスクロールしてしまうからです。

サブコマンドは次のような形式にして初めてサブコマンドとして（VTERMに）認識されます。



ヘッダ：

$１４

ＳＩ／ＳＯ　(sub inの時$０Ｆ，sub outの時$０Ｅ)

ＳＩで返って来るバイト数

ＳＯで送るバイト数

サブコマンド：

$００，$００が最初に付きます

エンド：$０３に固定

具体的な方法はリストを参照して下さい。また、"/Term"をオープンする方法は使わない方がよいでしょう。

```
PROCEDURE time_main
(*                               *)
(*    usage: time <on|off>.      *)
(*                               *)
base 0
param line: string[256]
```

```
dim buf(12): byte
dim path: byte; i: integer

    line = trim$( line )
    for i = 0 to 11
        read buf(i)
    next i
    if line = "" or line = "on" then
        run time_settime
        buf(7) = 1
    else if line = "off" then
        buf(7) = 0
    else
        print #2, "usage: time <on|off>."
        end
    endif ¥ endif
    put #1, buf
    end

    data $14, $0e, $00, $07
    data $00, $00, $40, $01, $06, $48, $00
    data $03
end


PROCEDURE time_settime
base 0
dim buf(14): byte
dim i: integer
dim tm: string[20]

    tm = date$
    for i = 0 to 13
        read buf(i)
    next i
    buf(9)  = val( mid$( tm, 10, 2 ) )
    buf(10) = val( mid$( tm, 13, 2 ) )
    buf(11) = val( mid$( tm, 16, 2 ) )
    put #1, buf
    end

    data $14, $0e, $00, $09
    data $00, $00, $3d, $02, $00
    data $00, $00, $00, $00
    data $03
end
```

## 3―2―5　at

atコマンドは指定した時刻に指定したコマンドを起動するものです、時刻はhh：mm：ssの形式でなければなりません。例えば、

```
OS9：at  12：00：00  echo  ＊＊＊＊  lunch  time  ＊＊＊＊  &
```

としておけば、12：00：00になると、

```
＊＊＊＊  lunch  time  ＊＊＊＊
```

というメッセージが現われることになります。ただし最後に'&'を付けないと、その時刻まで何もしないで待ち続けることになってしまいます。

このコマンドはsleepした後にコマンドを起動するという点で、次のものと同じです。

```
OS9：(sleep  4000；echo  ＊＊＊＊  lunch  time  ＊＊＊＊)&
```

この場合は、sleepするカウントを自分で計算しなければならない上、６５５３５ティック（すなわちＦＭ―７、８の場合で４３分、ＦＭ―１１の場合で１０分）しか待つことができません。

以上のことから、プログラムの中身の察しがついたのではないでしょうか。

まず第一パラメータとして時刻を置かなければならず、省略はできません。第二パラメータ以降は全体を一つのコマンドラインと見なします。これもまた省略できません。

指定した時刻と現在時刻を秒に換算して、さらにそれをティックに換算してその数値だけsleepすればよいわけです。現在時刻を知るにはＤＡＴＥ＄の右８文字がhh：mm：ssの形式になっている事を使います。

メインではgetargで第一パラメータを取り出します。それが指定した時刻であるはずで、残りがコマンドラインであるはずです。コマンドラインの中身がどうなっているか知る必要はありません。

時刻を秒に換算するのがat_gettimeです。この場合、秒は整数の範囲を越えるかもしれませんからＲＥＡＬ型にしておきます。

指定した時刻から現在時刻を差し引いた秒数だけsleepすればよいわけですがBasic09にはＳＬＥＥＰ文が無いので、代わりにプロシジャat_sleepを呼び出します、sleepはアセンブラで組んでおくと後あと便利ですが、ここではsleepコマンドを使用してBasic09で簡単に作っておきました。

sleepコマンドのパラメータは、秒単位ではなくティックを単位としていますので、at_sleepでは最初にその換算をします。この値もまたＲＥＡＬ型にしておく必要がある上、整数の上限である３２７６７を越えてもいけません。さもないとマイナスになってしまいます。　ティックに換算する時、ＦＭ―７、８とＦＭ―１１では１ティックの時間が異なります。リストではＦＭ―１１の場合を示していますので、ＦＭ―７、８で使用する場合はＴＩＣＫの初期値を１０００／４０＝２５にしておいて下さい。

３２７６７ティックを越える場合は３２７６７ティックごとにsleepし、それ以下の場合はその値だけsleepします。これを、残りがなくなるまで繰り返します。

この前後でsleepをlinkおよびunlinkしていますが、もしメモリ上にsleepが無いのであれば、linkをloadに変えて下さい。より一般的には次のようにすればよいでしょう。

```
on error goto 10
    shell "link sleep"
    goto 20
10 on error
    shell "load sleep"
20
```

以上ですが、ファイルのリストアウトの最中に指定時刻になったりすると、echoの表示程度では流されてしまう可能性があります。こういう事態を避ける一般的な方法はありませんので各人で工夫してみて下さい。

```
PROCEDURE at_main
base 0
 (*                                 *)
 (*     usage: at hh:mm:ss command  *)
 (*                                 *)
param argument: string[256]
dim time: integer
dim ctime, atime: real
dim arg: string[256]

    run getarg( argument, arg )
    run at_gettime( arg, atime )
    run at_gettime( right$( date$, 8 ), ctime )
    time = atime - ctime
    if time < 1 or argument = "" then
        print #2, "usage: at hh:mm:ss command"
        end
    endif
    run at_sleep( time )     ¥(* seconds *)
    shell argument
end



PROCEDURE at_gettime
base 0
param stm(8): byte
param time: real

    if stm(2) <> $3a or stm(5) <> $3a then
        time = -1
    else
        time = (stm(0)-$30)*36000+(stm(1)-$30)*3600+(stm(3)-$30)*600+(stm(4)-$3
0)*60+(stm(6)-$30)*10+(stm(7)-$30)
    endif
end



PROCEDURE at_sleep
param time: integer
dim rtimr: real
dim TICK, MAXINT: integer

    TICK = 1000/10
    MAXINT = 32767

    shell "link sleep"
    rtime = time
    rtime = rtime * TICK
    while rtime > 0 do
        if rtime > MAXINT then
            time = MAXINT
        else
            time = rtime
        endif
        shell "sleep " + str$( time )
        rtime = rtime - time
    endwhile
    shell "unlink sleep"
end
```

## 3－2－6 wild

これはワイルドカードを解釈するコマンドです。パラメータとして置くパスリストのどのレベルにワイルドカード文字を置いても構いません。

```
OS9：wild  *
```

でカレントディレクトリの全ファイルが表示され、

```
OS9：wild  /d0/usr/*/*.c
```

で/d0/usrの下にぶらさがっている全ディレクトリの中にあるCのソースファイルを全部表示します。

grepを思い出されたかもしれませんが、実際にやっていることはgrepとほぼ同じなのです。ただ違っているのは次の点です。

1．大文字と小文字を区別しない。
2．両端ともマッチしなければならない。
3．エスケープキャラクタは必要ない。

ＯＳ－９ではファイル名として大文字と小文字の区別をしません。無論ファイルを作る時には区別をしてディレクトリに書き込まれているわけですからファイル名をサーチしていく時にはどう出てくるかわかりませんが。

またgrepの時は、行の中のどこか一箇所でもパターンが含まれていればよかったのに対して、wildでは指定したファイル名の最初と最後が一致するもののみがマッチしたということになります。

```
OS9：wild  a*.c
```

という指定に対して、

```
yxzabc.c
abc.c9
```

というファイル名はマッチしません。

以上のように、マッチするべきものがファイル名であると限定されますと、だいぶgrepとは様子が変わってきます。エスケープキャラクタなどはもはや不要ですし、何より、パターンを整形しておくという手間が不要になります。

ではパターンマッチングの関数を挙げます。例によってプロシジャ名には接頭辞としてmatch_というものを付けてあります。これらはwildコマンドに組み込む時はwild_というふうに変えた方がよいでしょう。

```
PROCEDURE match_match
base 0
param file(32): byte; pat(256): byte
param s, p: integer
param match: boolean
dim c, c1, c2: integer

    while file(s) <> $FF and pat(p) <> $FF do
        match = TRUE
        c = pat(p) ¥ p = p + 1
        if c = $2A then     (* '*' *)
            if pat(p) = $FF then     (* only '*' *)
                end
            else
                while file(s) <> $FF do
                    run match_match( file, pat, s+0, p+0, match )
                    s = s + 1
                    if match then
                        end
                    endif
                endwhile
                match = FALSE
            endif
        else if c = $5B then     (* '[' *)
            run match_brakette( pat, p, file(s)+0, match )
        else if c = $5E then     (* '^' *)
            match = file(s) <> pat(p)
            p = p + 1
        else if c = $3F then     (* '?' *)
             (* already match *)
        else
            run match_toalower( c, c1 )
            run match_toalower( file(s)+0, c2 )
            match = ( c1 = c2 )
        endif ¥ endif ¥ endif ¥ endif
        if not match then
            end
        endif
        s = s + 1
    endwhile
    if file(s) = $FF and pat(p) = $FF then
        match = TRUE
    else
        match = FALSE
    endif
end


PROCEDURE match_brakette
base 0
param pat(256): byte
param p, c: integer
param match: boolean
dim found: boolean
dim c1, c2: integer

    run match_toalower( c+0, c )
    if pat(p) = $5E then     (* '^' *)
        match = FALSE
        p = p + 1
    else
        match = TRUE
    endif
    found = FALSE
    while pat(p) <> $FF and pat(p) <> $5D do     (* ']' *)
        if pat(p+1) = $2D and pat(p+2) <> $FF and pat(p+2) <> $5D then
            run match_toalower( pat(p)+0, c1 )
            run match_toalower( pat(p+2)+0, c2 )
            if c >= c1 and c <= c2 then
                found = TRUE
            endif
            p = p + 2
        else
            run match_toalower( pat(p)+0, c1 )
            if c = c1 then
                found = TRUE
            endif
        endif
        p = p + 1
    endwhile
    if pat(p) <> $FF then
        p = p + 1
    endif
    match = not ( match xor found )
end


PROCEDURE match_toalower
param c1, c2: integer

    if c1 >= $41 and c1 <= $5A then
        c2 = c1 + $20
    else
        c2 = c1
    endif
end
```

match_match へのパラメータは５つあります。ＢＯＯＬＥＡＮ型の match は値を返すためのものです。file と s でファイル名を表し、pat と p でパターンの文字列を表します。Ｃにおいては文字列に対するポインタ１本で済んだところを Basic09では文字列とそのインデックスという２つのもので表さなければなりません。そして一番最初に呼ぶ時は初期値として s と p に０を設定しておかなければなりません。ただ s と p の値は呼んだプロシジャに返す必要がないので値渡しにしてもよく、

```
RUN   match_match(   file,   pat,   0,   0,   match   )
```

で十分です。

else-if を頭を揃えて書いてあるのは case 文と同じだということを強調したいためです。メタキャラクタでなかった場合はファイル名とパターンの両方をいっ

たん小文字に直してから比較します。小文字に直す関数は備わっていませんのでプロシジャとして作っておきますが、関数にできないため値を持って返らせるための余計な変数が必要になります。それがc1およびc2です。

これでファイル名１つのパターンマッチが可能になりましたが、wildコマンドで指定するのは一般にパスリストだということを思い出して下さい。パスリストを何らかの方法で一つひとつのファイル名に分解し、各レベルごとにパターンマッチングを適用しなければなりません。またワイルドカードで指定したパスリストを、表示する時には正しいパスリストとして表示しなければならないことから、もとのパスリストとは別に正しいパスリストをどこかに保存しておかなければなりません。

一つのレベルに注目した時、一般にはワイルドカードで与えられますからマッチするものが２つ以上あり得ます、それらの一つひとつについてもまた２つ以上とマッチするかもしれません。１つのディレクトリについて、その下のレベルを全部調べた後で同じレベルの次のファイルについて調べる、ということになります。すなわちこれは再帰呼び出しそのものです。

一つのレベルからその下のレベルへ移る時はそのディレクトリへＣＨＤし、もどる時はＣＨＤ”..”とすればよいでしょう。その時は正しいパスリストとしてそのディレクトリ名を加えておきます。

メインプロシジャは例によって、パラメータの取出しのみ行い、それ以上のことは下位のプロシジャに委ねます。

さて問題はルートディレクトリです。何が問題かと言いますと、探索の方法がまったく違うのです。つまりルートディレクトリだけは、どのディレクトリの中身を探索しても存在しないのです。

ルートディレクトリとして何があるかを探索する方法としては、まずドライブテーブルを探索するするという手がありますが、手間がかかりすぎる事と、同一のディスクに２つ以上の名前が与えられている場合がある事から、ここでは避けることにします。かわりに存在しうるディスク名をＤＡＴＡ文で並べておき、順に試してみるという手を使います。もっともルートの指定にワイルドカードが使われていなければ、以上のような心配はまったく不要です。

ルートにワイルドカードが含まれていた場合はＤＡＴＡ文で持ってきたものとマッチングをとり、wild_wild を呼びます。

wild_wild に渡った時点で文字列変数 root には（もしあれば）ルートディレクトリ名が(正しく)入っています。wild_wild の中ではカレントディレクトリのみに注目していればよく、下のレベルを見る時は、再帰的に自分自身を呼び出します。その前後でＣＨＤを実行していることに注意して下さい。

なお、この中でカレントのパスリストを持っている pathlist の他に before という文字列を持っていますが、これはワイルドカードをマッチングさせる時の退避用です。つまり下のレベルを見る時はパスリストを現レベルまで作っておいてからでなければいけませんが、下のレベルは１つとは限らないため、１つが終ったらいったん元にもどし、再び次のレベルを試すというふうにしなければなりません。そのため、前のものを保存しておく必要があるのです。

ただしここでやっている方法は無駄が多く、スタックをやたらと使っているために、多重のワイルドカードを指定するとスタックオーバーフローを起こして実行を停止してしまいます。実際は、３段で駄目になるようです。つまり、

```
OS9：wild　/d0/＊/＊/＊.c
```

でＥＲＲＯＲ＃２０７を起こします。データエリアサイズを多めに指定しても、うまくいきません。しかし、wild_main そのものを起動するのであれば有効です。（３－２－９の flink を参照して下さい。）

```
OS9：wild_main(　"/d0/＊/＊/＊.c"　)　#20k
```

とすればだいじょうぶです。

しかしこれでは、コマンドとしての価値が半減してしまいます。対策としては、要するにローカルな変数を多く使用しないようにコーディングするということになります。before を使わずにすませるというのも一つの手ですし、ローカルな文字列を、次節で挙げる find のようにグローバルな変数にしてしまうのも効果的です。

```
PROCEDURE wild_main
base 0
param line: string[256]
dim pathlist: string[256]

    loop
        run getarg( line, pathlist )
        exitif pathlist = "" then
        endexit
        run wild_root( pathlist )
    endloop
end


PROCEDURE wild_root
base 0
param pathlist: string[256]
dim position: string[256]; root: string[256]; match, wild: boolean
dim i: integer

    run wild_separate( pathlist )
    if asc(pathlist) = $2F then
        run getarg( pathlist, root )
        run wild_iswild( root+"", wild )
        if wild then
            loop
                read position
                exitif position = "" then
                endexit
                run wild_match( position, root, 0, 0, match )
                if match then
                    on error goto 10
                        chd position
                        run wild_wild( pathlist+"", position )
                    10 on error
                endif
            endloop
        else
            on error goto 20
                chd root
                run wild_wild( pathlist, root )
            20 on error
        endif
    else
        run wild_wild( pathlist+"", "" )
    endif
end
    data "/r0", "/d0", "/nd0", "/nd1", "/md0", "/md1"
    data ""
end


PROCEDURE wild_separate
base 0
param pathlist(256): byte
dim i: integer

    for i = 1 to 255    ¥(* because pathlist(0) = '/' if root *)
        exitif pathlist(i) = $FF then
        endexit
        if pathlist(i) = $2F then
            pathlist(i) = $20
        endif
    next i
end


PROCEDURE wild_iswild
base 0
param file(256): byte; wild: boolean
dim i: integer
```

```
    wild = FALSE
    for i = 0 to 255
        exitif file(i) = $FF then
        endexit
        exitif file(i)=$2A or file(i)=$3F or file(i)=$5B or file(i)=$5E then
            wild = TRUE
        endexit
    next i
end


PROCEDURE wild_wild
base 0
param pathlist, before: string[256]
dim pattern,position: string[256]; file: string[32]; path: byte; match: boolean

    if pathlist = "" then
        print before
        end
    endif

    run getarg( pathlist, pattern )
    open #path, ".": dir+read
    get #path, file
    get #path, file
    while not eof(#path) do
        get #path, file
        if asc(file) <> 0 then
            run wild_shapename( file )
            run wild_match( file, pattern, 0, 0, match )
            if match then
                if before = "" then
                    position = file
                else
                    position = before + "/" + file
                endif
                if pathlist = "" then
                    print position
                else
                    on error goto 10
                        chd file
                        run wild_wild( pathlist+"", position+"" )
                        chd ".."
                    10 on error
                endif
            endif
        endif
    endwhile
    close #path
end


PROCEDURE wild_shapename
base 0
param file(32): byte
dim i: integer

    for i = 0 to 28
        if file(i) > $7F then
            file(i) = land( file(i), $7F )
            file(i+1) = $FF
            end
        endif
    next i
    file(29) = $FF
end
```

## 3—2—7　find

wildの応用として、ファイルをディレクトリ中から探し出すコマンドであるfindを作ります。

findはあるディレクトリから下の全ディレクトリを辿って、ワイルドカードを含むファイル名で指定したファイルを探します。ただし指定するファイル名はパスリストであっては（間に'／'を含んでは）なりません。

なお探索開始のディレクトリを指定しなければ、カレントディレクトリから始めます。開始ディレクトリの指定は'－r＝'の次にディレクトリ名を置いたものです。また通常は該当するファイルを1つ見つけた時点で終了しますが、見つかっても見つからなくても全部を探索するというオプションも付けます。これが'－a'です。

```
OS9：find　－r＝/d0/usr　＊.c
```

とすれば/d0/usrディレクトリの下のすべてのファイルを探索し、Cのソースファイルが1つでも見つかった時点でそのパスリストを表示して終了します。

```
OS9：find　－r＝/d0/usr　－a　＊.c
```

とすると、Cのソースファイルをすべて表示します。

探索の途中でパーミッションが合わなくて読めなかったものは無視します。また一つも見つからなかった場合はエラーではなく、何も表示せず終了します。

リストを見て気付いたかもしれませんが、やっていることはwildとほとんど同じです。

変数のいくつかをグローバル変数のように取り扱ってみました。GLOVALという型に宣言されたexternという構造体がそれで、オプション'－a'のフラグallおよび'－r＝'の値であるrootもこの中に入れてあります。doneは探索が終了したかどうかのフラグです。もし、'－a'オプションが無かった場合はファイルを1つ見つけた時点で終了しなければなりませんが、再帰の途中で終了しても終了したという事実を上位のルーチンにまで返してやらなければ終了できません。

find_find を呼び出したルーチンはこの done の値を見て以後探索を続けるかどうか判断します。

現在どこのディレクトリにいるかは extern. plist 配列とそのインデックス n で表します。これは wild の場合もそうでしたが、マッチするファイル名を出力する際に、パスリストとして出力するためのものです。

下のディレクトリに移る時は、plist にそのディレクトリ名を付け加えた上でそのディレクトリにＣＨＤし、n を１つ増やして find_find を呼び出します。帰ってきたらＣＨＤ　”..”でもとのレベルにもどります。この時フラグ done がＴＲＵＥだったならば、すでに１つ見つかっておりかつ’－ a ’の指定が無かったということなので、ただちにリターンします。

また下のディレクトリに移る際にはエラートラップ（ON　ERROR　GOTO文）をかけておき、パーミッションが合わなかった場合はそのディレクトリを無視するようにしています。

パターンマッチングは前にも出てきた match_match を使用します。find と組み合わせるためにプロシジャ名を find_match のように find_の接頭辞を付けて下さい。

```
PROCEDURE find_main
base 0
type GLOVAL = pattern, root, plist(13): string[32]; all, done: boolean
param argument: string[256]
dim extern: GLOVAL; arg: string[256]; opterr: boolean

    extern.all = FALSE
    extern.done = FALSE
    extern.root = ""
    extern.pattern = ""
    run find_option( extern, argument+"", opterr )
    if opterr then
        print #2, "usage: find <-r=start_directory> <-a> pattern."
        end
    endif
    loop
        run getarg( argument, arg )
        exitif arg = "" then
        endexit
        if asc(arg) <> $2D then     (* '-' *)
            extern.pattern = arg
            if extern.root <> "" then
                chd extern.root
            endif
            run find_find( extern, 0 )
        endif
    endloop
end


PROCEDURE find_option
base 0
type GLOVAL = pattern, root, plist(13): string[32]; all, done: boolean
param extern: GLOVAL; argument: string[256]; opterr: boolean
dim arg: string[256]; n: integer

    opterr = FALSE
    loop
        run getarg( argument, arg )
        exitif arg = "" then
        endexit
        if arg = "-a" or arg = "-A" then
            extern.all = TRUE
        else if left$( arg, 3 ) = "-r=" then
            extern.root = right$( arg, len(arg)-3 )
        else
            if asc(arg) = $2D then
                opterr = TRUE
            endif
        endif ¥ endif
    endloop
end


PROCEDURE find_find
base 0
type GLOVAL = pattern, root, plist(13): string[32]; all, done: boolean
param extern: GLOVAL; n: integer
dim path: byte; dr(32): byte; match: boolean

    open #path, ".": dir+read
    get #path, dr
    get #path, dr
    while not eof( #path ) do
        get #path, dr
        if dr(0) <> 0 then
            run find_shapename( dr, extern.plist(n) )
            run find_match( extern.plist(n), extern.pattern, 0, 0, match )
            if match then
                if extern.root <> "" then
                    print extern.root; "/";
                endif
                for i = 0 to n-1
                    print extern.plist(i); "/";
                next i
                print extern.plist(n)
                if not extern.all then
                    extern.done = TRUE
                    end
                endif
            endif
            on error goto 10
                chd extern.plist(n)
                run find_find( extern, n+1 )
                chd ".."
                exitif extern.done then
                    end
                endexit
            10 on error
        endif
    endwhile
    close #path
end


PROCEDURE find_shapename
base 0
param dr(32): byte
param name(32): byte
dim i: integer

    for i = 0 to 28
        exitif dr(i) > $7F then
            name(i) = land( dr(i), $7F )
            name(i+1) = $FF
        endexit
        name(i) = dr(i)
    next i
end


PROCEDURE find_match
base 0
param file(32): byte; pat(256): byte
param s, p: integer
param match: boolean
dim c, c1, c2: integer

    while file(s) <> $FF and pat(p) <> $FF do
        match = TRUE
        c = pat(p) ¥ p = p + 1
        if c = $2A then    (* '*' *)
            if pat(p) = $FF then    (* only '*' *)
                end
            else
                while file(s) <> $FF do
                    run find_match( file, pat, s+0, p+0, match )
                    s = s + 1
                    if match then
                        end
                    endif
                endwhile
                match = FALSE
            endif
        else if c = $5B then    (* '[' *)
            run find_brakette( pat, p, file(s)+0, match )
        else if c = $5E then    (* '^' *)
            match = file(s) <> pat(p)
            p = p + 1
        else if c = $3F then    (* '?' *)
            (* already match *)
        else
            run find_toalower( c, c1 )
            run find_toalower( file(s)+0, c2 )
            match = ( c1 = c2 )
        endif ¥ endif ¥ endif ¥ endif
        if not match then
            end
        endif
        s = s + 1
    endwhile
    if file(s) = $FF and pat(p) = $FF then
        match = TRUE
    else
        match = FALSE
    endif
end


PROCEDURE find_brakette
base 0
param pat(256): byte
param p, c: integer
param match: boolean
dim found: boolean
dim c1, c2: integer

    run find_toalower( c+0, c )
    if pat(p) = $5E then    (* '^' *)
        match = FALSE
        p = p + 1
    else
        match = TRUE
    endif
    found = FALSE
    while pat(p) <> $FF and pat(p) <> $5D do    (* ']' *)
        if pat(p+1) = $2D and pat(p+2) <> $FF and pat(p+2) <> $5D then
            run find_toalower( pat(p)+0, c1 )
            run find_toalower( pat(p+2)+0, c2 )
            if c >= c1 and c <= c2 then
                found = TRUE
            endif
            p = p + 2
        else
            run find_toalower( pat(p)+0, c1 )
            if c = c1 then
                found = TRUE
            endif
        endif
        p = p + 1
    endwhile
    if pat(p) <> $FF then
        p = p + 1
    endif
    match = not ( match xor found )
end


PROCEDURE find_toalower
param c1, c2: integer

    if c1 >= $41 and c1 <= $5A then
        c2 = c1 + $20
    else
        c2 = c1
    endif
end
```

## 3—2—8　smem

このプログラムは、ファイル中の指定された１６進データを見つけてそのファイル中のアドレスを出力するものです。見つけるデータは、16進２桁の文字列をスペースで区切って並べて指定します。例えば、”ＡＢ”を見つけたいときは、

```
smem  41  42
      “A“   “B“  のＡＳＣＩＩコード（１６進）
```

とします。使い方は、

```
smem  41  42  <（ファイル名）
```

とするか、または、

```
list （ファイル名） ！ smem  41  42
```

とパイプを利用します。

### プログラムの説明

まず、見つけたいデータを１６進が並んだ形から文字列に直します。文字列変数 pattern に16進データが入っているので、これを getarg によって２桁ごとに区切り、そのアスキーコードの文字に直し、あわせて１つの文字列（patchr、長さ length）にします。次に、ファイルの最初から１バイトずつ、patchr の最初の文字と一致するまで読み込みます。一致したら、(length—1) 文字を読み込み、あわせて長さ length の文字列 (match) に代入します。そして、match と patchr を比較し、一致したならその文字列のあった先頭アドレスを表示して次の文字から同様にサーチします。もし一致しなければ、次の１バイトを読み込み、文字列 match の最後につけ、先頭の１バイトを取りのぞいたものを新しい match として、patchr との比較を繰り返します。

変数 adr はつねに match の先頭アドレスをさしています。見つかったときは、adr を１６進で出力します。(prthex)。

```
PROCEDURE smem
param argument: string[256]
dim i, j, length, adr, dif: integer
dim dat, c: string[1]
dim patchr, match: string[100]
dim dread: string[2]
dim pat: byte

    on error goto 100
    patchr = ""
    match = ""
    i = 1
    loop
        run getarg( argument, dread )
        exitif dread="" then
        endexit
        patchr = patchr+chr$(val("$"+dread))
        i = i + 1
    endloop
    length = i - 1
    print patchr
    print
    adr = 0
    loop
        loop
            get #0, dat
            exitif dat = left$(patchr,1) then
            endexit
            adr = adr + 1
        endloop
        match = ""
        for i = 1 to length-1
            get #0, c
            match = match + c
        next i
        match = dat + match
        loop
            exitif match = patchr then
                run prthex( adr )
                adr = adr + length
            endexit
            get #0, c
            match = right$(match,length-1)+c
            adr = adr + 1
        endloop
    endloop
    end

100
    if err = 211 then
      print "end of file"
    else
      error err
    endif
end


PROCEDURE prthex
param adr: integer
dim hex(6): string[1]
dim i(6), temp: integer
dim str: string[6]

    j = 6
    temp = adr
    repeat
        i(j) = temp-int(temp/16)*16
        temp = int(temp/16)
        j = j - 1
    until temp = 0
    str = ""
    for k = j + 1 to 6
        if i(k) < 10 then
            hex(k) = chr$($30+i(k))
        else
            hex(k) = chr$(55+i(k))
        endif
        str = str + hex(k)
    next █
    print
    print "address = "; right$( "      "+str, 6 )
end
```

## 3—2—9　flink

F—BASIC用のリンカープログラムです。

ソースはF—BASICに限らず、行番号付きの一般的なBASICのものならばそのまま使用可能です。このプログラムは、F—BASICから Basic09へのプログラムの移植を容易にするために作りました。

F—BASICでは行番号はプログラムの一部であると考えているため、サブルーチンごとにファイルに分けておいて後で組み合わせるといったことが非常に難しく、プログラムの蓄積はほとんど不可能です。マージしようとするのならそれぞれのプログラムの行番号が重複しないように注意を払わなければなりませんし、GOSUBなどで他のファイルのプログラムに飛ぼうとするにはプログラムの書き換えが必要です。

flink は各プログラムファイルの行番号の付け換えを行います。また、GOSUBの飛び先として、行番号ではなくファイル名を指定できるようにしてあります。飛び先としてファイル名を指定した場合は、そのファイルの第1行目を実際の飛び先として適当な行番号に置き換えます。

例えば次の5つのファイルがあるとします。

```
OS9:list main
100 '*
110 '**** sample for 'flink' ****
120 '*
130 CLS
140 '-----------------------------------
150 WHILE 1
160     A$=INKEY$
170     IF A$="1" THEN GOSUB "left":  GOTO 230
180     IF A$="2" THEN GOSUB "down":  GOTO 230
190     IF A$="3" THEN GOSUB "right": GOTO 230
200     IF A$="5" THEN GOSUB "up":    GOTO 230
210     IF A$="9" THEN 240
220     IF RND > 0.5 THEN PRINT "*"; CHR$(&h1D); ELSE PRINT "@"; CHR$(&h1D);
```

```
230 WEND
240 '------------------------------------
250 END
260 '

OS9:list right
100 '** move right **
110 PRINT CHR$(&H1C);
120 RETURN
130 '

OS9:list left
100 '** move left **
110 PRINT CHR$(&H1D);
120 RETURN
130 '

OS9:list up
100 '** move up **
110 PRINT CHR$(&H1E);
120 RETURN
130 '

OS9:list down
100 '** move down **
110 PRINT CHR$(&H1F);
120 RETURN
130 '
```

これをリンクするためにはまず、メインとなるファイルを一つ指定します。この場合は test です。flink を起動します。

```
OS9：flink　main
```

これによって main 内で参照されているファイルを自動的に取り込み、もしそのファイルの中で参照しているファイルがあれば取り込み、というふうに再帰的に処理します。最終的には次のような１本のプログラムとして出力されます。

```
1000 '*
1010 '**** sample for 'flink' ****
1020 '*
1030 CLS
1040 '-------------------------------------
1050 WHILE 1
1060     A$=INKEY$ .
1070     IF A$="1" THEN GOSUB 1170:  GOTO 1130
1080     IF A$="2" THEN GOSUB 1210:  GOTO 1130
1090     IF A$="3" THEN GOSUB 1250: GOTO 1130
1100     IF A$="5" THEN GOSUB 1290:    GOTO 1130
1110     IF A$="9" THEN 1140
1120     IF RND > 0.5 THEN PRINT "*"; CHR$(&h1D); ELSE PRINT "@"; CHR$(&h1D);
1130 WEND
1140 '-------------------------------------
1150 END
1160 '
1170 '** move left **
1180 PRINT CHR$(&H1D);
1190 RETURN
1200 '
1210 '** move down **
1220 PRINT CHR$(&H1F);
1230 RETURN
```

```
1240 '
1250 '** move right **
1260 PRINT CHR$(&H1C);
1270 RETURN
1280 '
1290 '** move up **
1300 PRINT CHR$(&H1E);
1310 RETURN
1320 '
```

もし２箇所以上から同じファイルが参照された時は、１つだけがマージされて同じ場所を指すことになります。

一応ＧＯＳＵＢだけにファイルの参照を許すことにしてありますが、リストのflink_pass1中の

```
if word="GOSUB" then
```

を

```
if word="GOSUB" or word="GOTO" or word="THEN"
                                   or word="ELSE" then
```

とすればＧＯＳＵＢ以外にもファイル名の指定が可能になります。

なお flink はプログラムの内容をいっさいチェックしません。ＲＥＴＵＲＮやＥＮＤがあるかどうかも調べませんのでプログラマの側でチェックして下さい。また flink は完璧さを求めることはしませんでした。単純に、ＧＯＴＯやＧＯＳＵＢの直後の行番号またはダブルクォートで囲まれたファイル名についてだけ置き換えを行い、たとえ文法的な間違いがあったとしても関知しません。

さて、flink はこのままＰＡＣＫしてもメモリフルエラーが発生して実行することができません。wild の時もそうでしたがローカル変数を多用するとすぐにスタックがオーバーフローしてしまうのです。直接 flink_main を起動するのならデータエリアサイズを広げることができます。

```
OS9：runb　#8k　flink_main(　"test"　)
```

しかし、これではあまりに使いにくいので、章の冒頭で挙げた module. a を少し手直ししましょう。

行８３の clrb を、希望するデータエリアのサイズをロードするように換えま

す。Ｂレジスタに入れるのはページ単位の値ですから間違えないようにして下さい。下の例では８ＫＢのデータエリアを確保しています。データエリアをもっと多く取りたいのであれば、いくつでも好きな値に書き換えて下さい。

```
00077              *
00078              *     U reg. means parameter address
00079              *     Y reg. means parameter size
00080              *
00081    004F 308DFFCD               leax  Runb,pcr   name of Runb
00082    0053 8611                   lda   #Prgrm+Objct
00083    0055 C620                   ldb   #32        data size #8K
00084    0057 103F03                 os9   F$Fork
00085    005A 2506                   bcs   exit
00086    005C 103F04                 os9   F$Wait
00087    005F 2501                   bcs   exit
00088    0061 5F                     clrb
00089    0062 103F06      exit       os9   F$Exit
00090
```

さてflinkにはオプションとして次のものを付けました。”−s=”で、出力するリストの行番号のスタート値を指定します。また”−d=”で、行番号の増分値を指定します。これらのデフォルト値は１０００から１０刻みとなっていますが、flink_optionの最初の値を変更すればデフォルト値を変えられます。

```
PROCEDURE flink_main
base 0
 (*                                 *)
 (*    usage: flink <opt> file <opt>.    *)
 (*      opt: -s=1000  start line number *)
 (*           -d=10    step             *)
 (*                                 *)
param argument: string[256]
type GLOVAL = name(32): string[16]; lin(32), stt(32): integer; fdp: integer
dim extern: GLOVAL; ln(500), lp: integer; buf(512): byte; bp: integer
dim f$: string[16]; opath: byte
dim start, stp: integer
dim arg: string[256]
dim opterr: boolean

    run flink_option( extern, argument+"", start, stp, opterr )
    if opterr then
        run flink_usage
        end
    endif

    extern.name(0) = ""
    loop
        run getarg( argument, arg )
        exitif arg = "" then
        endexit
        exitif asc( arg ) <> $2d then
            extern.name(0) = arg
        endexit
    endloop
    if extern.name(0) = "" then
        run flink_usage
        end
    endif

    bp = 0
    extern.fdp = 1
    run flink_pass1( extern.name(0), extern.lin(0), extern, buf, bp )

    for i = 0 to extern.fdp-1
        extern.stt(i) = start
        start = start + stp * extern.lin(i)
    next i

    for i = 0 to extern.fdp-1
        run flink_pass2(extern.name(i), extern.stt(i), stp, buf, bp, ln, lp)
        run flink_pass3(extern.stt(i),stp,extern,extern.name(i),buf,bp,ln,lp)
    next i
end


PROCEDURE flink_option
base 0
type GLOVAL = name(32): string[16]; lin(32), stt(32): integer; fdp: integer
param extern:GLOVAL; argument:string[256]; start, stp:integer; opterr:boolean
dim arg: string[256]

    opterr = FALSE
    start = 1000
    stp = 10
    loop
        run getarg( argument, arg )
        exitif arg = "" then
        endexit
        if asc( arg ) = $2d then
            if left$( arg, 3 ) = "-s=" then
                start = val( right$( arg, len(arg)-3 ) )
            else if left$( arg, 3 ) = "-d=" then
                stp = val( right$( arg, len(arg)-3 ) )
            else
                opterr = TRUE
                end
            endif ¥ endif
        endif
    endloop
end


PROCEDURE flink_usage

    print #2, "usage: flink <opt> file <opt>."
    print #2, "  opt: -s=1000  start line number."
    print #2, "       -d=10    step."
end



PROCEDURE flink_pass1
base 0
type GLOVAL = name(32): string[16]; lin(32), stt(32): integer; fdp: integer
param f$: string[16]; ln: integer; extern: GLOVAL; buf(512): byte; bp: integer
dim path: byte; word: string[200]; tp: string[1]; n: integer; exist: boolean
dim errnum: integer

    on error goto 10
    open #path, f$: read
    on error

    ln = 0
    loop
        run flink_getword( path, word, tp, buf, bp )
        exitif tp = chr$($ff) then              (* eof *)
        endexit
        while tp = " " do
            run flink_getword( path, word, tp, buf, bp )
        endwhile
        exitif tp <> "0" then
            print #2, "line number not exist !   line "; ln; " in "; f$
        endexit
        loop
            run flink_getword( path, word, tp, buf, bp )
            exitif tp = chr$(13) then
                ln = ln + 1
            endexit
            if tp = "A" then
                run flink_upper( word )
                if word = "GOSUB" then
                    repeat
                        run flink_getword( path, word, tp, buf, bp )
                    until tp <> " "
                    if tp = chr$($22) then
                        run flink_fdsearch( extern, word, n )
                        if n = -1 then
                            n = extern.fdp
                            extern.name(n) = word
                            extern.fdp = extern.fdp + 1
                            run flink_pass1( word,extern.lin(n),extern,buf,bp )
                        endif
                    endif
                endif
            endif
        endloop
    endloop
    close #path
    end

10
    on error
    errnum = err
    if errnum = 216 then
        print #2, "can't open "; f$; ". in pass1."
    endif
    error errnum
end



PROCEDURE flink_pass2
base 0
param f$: string[16]; start, stp: integer; buf(512); bp: integer
param ln(500), lp: integer
dim path: byte; word: string[200]; tp: string[1]; c: integer
dim errnum: integer

    on error goto 10
    open #path, f$: read
    on error

    lp = 0
    loop
        run flink_getword( path, word, tp, buf, bp )
        exitif tp = chr$($ff) then
        endexit
        ln(lp) = val(word) ¥ lp = lp + 1
        repeat
            run flink_fgetc( path, c, buf, bp )
        until c = $0d
    endloop
    close #path
    end

10
    on error
    errnum = err
    if errnum = 216 then
        print #2, "can't open "; f$; ". in pass2."
    endif
    error errnum
end
```

```
PROCEDURE flink_pass3
base 0
type GLOVAL = name(32): string[16]; lin(32), stt(32): integer; fdp: integer
param start, stp: integer
param extern: GLOVAL; f$: string[16]; buf(512): byte; bp: integer
param ln(500), lp: integer
dim path: byte; word: string[200]; tp: string[1]
dim l, n, i: integer; exist: boolean
dim errnum: integer

    on error goto 10
    open #path, f$: read
    on error

    l = 0
loop
    run flink_getword( path, word, tp, buf, bp )
    exitif tp = chr$($ff) then
    endexit
    if tp = " " then
        print word;
        run flink_getword( path, word, tp, buf, bp )
    endif
    print start + l * stp;
    loop
        run flink_getword( path, word, tp, buf, bp )
        exitif tp = chr$(13) then
            print
            l = l + 1
        endexit
        if tp = chr$($22) then
            print chr$($22); word; chr$($22);
        else if tp <> "A" then
            print word;
        else
            print word;
            run flink_upper( word )
            while word="GOTO"or word="GOSUB"or word="THEN"or word="ELSE"do
                run flink_getword( path, word, tp, buf, bp )
                if tp = " " then
                    print word;
                    run flink_getword( path, word, tp, buf, bp )
                endif
                exitif tp = "0" then
                    n = val( word )
                    exist = false
                    for i = 0 to lp-1
                        exitif ln(i) = n then
                            exist = true
                        endexit
                    next i
                    if exist then
                        print start + i * stp;
                    else
                        print #2, "undefined line number !!"
                        print word;
                    endif
                endexit
                exitif tp = chr$($22) then
                    run flink_fdsearch( extern, word, n )
                    if n = -1 then             (* not exist *)
                        print chr$($22); word; chr$($22);
                    else
                        print extern.stt(n);
                    endif
                    endexit
                    exitif tp <> "A" then
                        print word;
                    endexit
                    print word;
                    run flink_upper( word )
                endwhile
            endif ¥ endif
        endloop
        exitif tp = chr$($ff) then
        endexit
    endloop
    close #path
    end

10
    on error
    errnum = err
    if errnum = 216 then
        print #2, "can't open "; f$; ". in pass3."
    endif
    error errnum
end


PROCEDURE flink_fdsearch
base 0
type GLOVAL = name(32): string[16]; lin(32), stt(32): integer; fdp: integer
param extern: GLOVAL; word: string[200]; n: integer
dim i: integer

    n = -1
    for i = 0 to extern.fdp-1
        exitif extern.name(i) = word then
            n = i
        endexit
    next i
end



PROCEDURE flink_getword
base 0
param path: byte; word: string[200]; tp: string[1]
param buf(512): byte; bp: integer
dim c:integer

    run flink_fgetc( path, c, buf, bp )
    loop
        exitif c = -1 then         (* eof *)
            tp = chr$($ff)
            word = ""
        endexit
        exitif c = $20 then
            tp = " "
            word = chr$(c)
            loop
                run flink_fgetc( path, c, buf, bp )
                exitif c <> $20 then
                    run flink_putbak( c, buf, bp )
                endexit
                word = word + chr$(c)
            endloop
        endexit
        exitif c >= $30 and c <= $39 then
            tp = "0"
            word = chr$(c)
            loop
                run flink_fgetc( path, c, buf, bp )
                exitif c < $30 or c > $39 then
                    run flink_putbak( c, buf, bp )
                endexit
                word = word + chr$(c)
            endloop
        endexit
        exitif c >= $41 and c <= $5a or c >= $61 and c <= $7a then
            tp = "A"
            word = chr$(c)
            loop
                run flink_fgetc( path, c, buf, bp )
                exitif ( c < $41 or c > $5a ) and ( c < $61 or c > $7a ) then
                    run flink_putbak( c, buf, bp )
                endexit
                word = word + chr$(c)
            endloop
        endexit
        exitif c = $22 then
            tp = chr$(c)
            word = ""
            loop
                run flink_fgetc( path, c, buf, bp )
                exitif c = $0d then
                    run flink_putbak( c, buf, bp )
                endexit
                exitif c = $22 then
                endexit
                word = word + chr$(c)
            endloop
        endexit
        exitif true then
            tp = chr$(c)
            word = chr$(c)
        endexit
    endloop
end


PROCEDURE flink_upper
base 0
param word(200): byte
dim i: integer

    for i = 0 to 199
        exitif word(i) = $ff then
        endexit
        if word(i) >= $61 and word(i) <= $7a then
            word(i) = word(i) - $20
        endif
    next i
end


PROCEDURE flink_fgetc
base 0
param path: byte; c: integer; buf(200): byte; bp: integer
dim s: string[1]
dim errnum: integer

    on error goto 999

    if bp > 0 then
        bp = bp - 1 ¥ c = buf(bp)
    else
        get #path, s ¥ c = asc(s)
    endif
    end

999
    errnum = err
    if errnum = 211 then      (* eof *)
        c = -1
    else
        error errnum
    endif
end


PROCEDURE flink_putbak
base 0
param c: integer; buf(200): byte; bp: integer

    buf(bp) = c ¥ bp = bp + 1
end
```

## 3－2－10　bprep

bprepはflinkの中で使用していたPROCEDUREであるgetwordを用いた、ごく簡単なプリプロセッサです。現在のところ2つの機能しか持ちあわせていません。Cに倣って'#'で始まる行をbprepに対する指令とみなします。

#include ファイル名

は、この行の位置に、指定したファイルを取り組みます。

#define 文字列1　文字列2

は、これ以下の文中で、文字列1を文字列2に置換します。

bprepにはBasic09を使い易くするという意図があり、とりあえず最も簡単な機能だけ持たせてみました。ただし、現在のところ引数付きのdefineはサポートしていません。

```
PROCEDURE bprep_main
 (*                          *)
 (*    usage: bprep file.    *)
 (*                          *)
base 0
param file: string[256]
dim beffor(200), after(200): string[32]; p: integer
dim word: string[200]

    p = 0
    run prep( file, beffor, after, p, word )

end


PROCEDURE bprep_prep
base 0
param file: string
param beffor(200), after(200): string[32]; p: integer
param word: string[200]
dim buffer(100): byte; bp: integer
dim tp: string[1]
dim path: byte

    open #path,file:read
    bp=0

    loop
        run bprep_getword( path, word, tp, buffer, bp )
        exitif tp = chr$($ff) then
        endexit
        repeat
            exitif tp = chr$($0d) then
                print word;
            endexit
            exitif word <> "#" then
                if tp = chr$($22) then
                    print tp; word; tp;
                else
                    run bprep_trans( word, beffor, after, p )
                    print word;
                endif
                repeat
                    run bprep_getword( path, word, tp, buffer, bp )
                    if tp = chr$($22) then
                        print tp; word; tp;
                    else
                        run bprep_trans( word, beffor, after, p )
                        print word;
                    endif
                until tp=chr$($0d)
            endexit
            run bprep_getword( path, word, tp, buffer, bp )
            run bprep_upper( word )
            exitif word = "difine" then
                run bprep_getword( patn, word, tp, buffer, bp )
                if tp <> " " then
                    print #2, "error !"
                endif
                run bprep_getword( path, beffor(p), tp, buffer, bp )
                run bprep_getword( path, word, tp, buffer, bp )
                if tp <> " " then
                    print #2, "error !"
                endif
                run bprep_getword( path, after(p), tp, buffer, bp )
                p = p + 1
            endexit
            exitif word = "include" then
                run bprep_getword( path, word, tp, buffer, bp )
                if tp <> " " then
                    print #2, "error !"
                endif
                run bprep_getword( path, word, tp, buffer, bp )
                if tp = chr$($22) then
                    rem run bprep_getword( path, word, tp, buffer, bp )
                endif
                file = word
                run bprep_prep( file, beffor, after, p, word )
            endexit
            repeat
                run bprep_getword( path, word, tp, buffer, bp )
            until tp = chr$($0d) or tp = chr$($ff)
        until true
    endloop
end


PROCEDURE bprep_trans
base 0
param word: string[200]
param beffor(200), after(200): string[32]; p: integer
dim i: integer

    for i = 0 to p-1
        exitif beffor(i) = word then
            word = after(i)
        endexit
    next i
end
```

# 3－3　グラフィック

## 3－3－1　gcursor

ＦＭシリーズはグラフィックをサブＣＰＵが担当しており、適当なコマンドを送るだけで比較的簡単にグラフィックが実現できます。ＦＭシリーズのＯＳ－９ではこれらのうちのいくつかがBasic09のサブルーチンモジュールとして用意されています。ところが、gcursorはまだサポートされていません。これはgcursorを実行中はサブシステムが（最悪の場合は半永久的に）占領されてしまうという危険があるためだと思われます。

これさえ覚悟すればgcursorを実行しても構わないでしょう。

さてgcursorのサブコマンドは次のようになっています。

サブに送るデータ

| | | | | | | |
|---|---|---|---|---|---|---|
| ＄００ | ＄００ | ＄１Ｆ | ＣＬ | Ｎ | Ｘ | Ｙ |

サブから受け取るデータ

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ｅｒｒ | ／ | ／ | Ｘ１ | Ｙ１ | Ｘ２ | Ｙ２ | … | Ｘｎ | Ｙｎ |

さて、gcursorを実行する際にはカーソルが邪魔ですので、同様にサブコマンドを使ってカーソルのオンオフを制御します。こちらの方は単に送るだけです。

これは実はdisplayコマンドで実行できます。例えばＦＭ－７、８では、

```
OS9：display  14  0e  00  04  00  00  0c  02  03
```

としますと、カーソルが消えます。($0Cはコンソールを制御するサブコマンドです。)

上の例はそのままでShellプロシジャとして使用可能です。ただし、サブコマンドは機種によって違います。上の例ではＦＭ—１１の場合に正常な動作は保証されず、次のようにしなければなりません。

OS9：display 14 0e 00 05 00 00 0c 00 42 03

ＦＭ—７、８の場合

| | | | |
|---|---|---|---|
| ＄００ | ＄００ | ＄０Ｃ | ＄０２ |

ＦＭ—１１の場合

| | | | | |
|---|---|---|---|---|
| ＄００ | ＄００ | ＄０Ｃ | ＄００ | ＄４２ |

さて、以上のことを踏まえた上でgcursorを書いてみましょう。

```
     *****************************************
     *
     *    gcursor  --  graphic cursor
     *
     *    usage: RUN gcursor( x, y, x1, y1, ... xn, yn )
     *

                          ifp1
                          endc

0002             LEBEL    set    2

0000 87CD0100             mod    Size,Name,Sbrtn+Objct,ReEnt+1,Entry,Mem
0080             Mem      equ    128
000D 47437572    Name     fcs    /GCursor/
0014 01                   fcb    1          edition number

                          ifeq   LEBEL-1
                          else
0015 140E0500    curon    fcb    $14,$0e,$05,$00,$00,$0c,$00,$43,$03
001E 140E0500    curof    fcb    $14,$0e,$05,$00,$00,$0c,$00,$42,$03
     *
     * if your system is lebel-1, please change
     *
     *  curon fcb $14,$0e,$04,$00,$00,$0c,$03,$03
     *  curof fcb $14,$0e,$04,$00,$00,$0c,$02,$03
     *
                          endc

0000             STDIN    equ    0
0001             STDOUT   equ    1
0038             E$PAR    equ    56
```

```

0027                Entry    equ    *
0027 33E4                    leau   ,s            flame pointer
0029 EC42                    ldd    2,u           get parameter count
002B 10830002                cmpd   #2
002F 2531                    blo    parerr        parameters too short
0031 10830016                cmpd   #22
0035 222B                    bhi    parerr        parameters too many
0037 54                      lsrb
0038 2528                    bcs    parerr        parameter number is odd
003A EC46                    ldd    6,u
003C 10830002                cmpd   #2
0040 2620                    bne    parerr
0042 EC4A                    ldd    10,u
0044 10830002                cmpd   #2
0048 2618                    bne    parerr

004A 32E8D5                  leas   -43,s         keep work area
004D 8D28                    bsr    offcurs       cursor off
004F 1F41                    tfr    s,x           buffer address
0051 8D35                    bsr    putcom        put sub-command for gcursor
0053 2506                    bcs    gcurs1
0055 8D71                    bsr    getcom        get return values
0057 2502                    bcs    gcurs1
0059 8D7D                    bsr    setdat
005B 8D09           gcurs1   bsr    oncurs        cursor on
005D 32E82B                  leas   43,s
0060 5F                      clrb
0061 39                      rts

0062                parerr   equ    *
0062 53                      comb
0063 C638                    ldb    #E$PAR        parameter error
0065 39                      rts

          ******************************************
          *
          *     cursor on
          *

0066                oncurs   equ    *
0066 3436                    pshs   d,x,y
0068 308DFFA9                leax   curon,pcr
006C 108E0009                ldy    #curof-curon
0070 8601                    lda    #STDOUT
0072 103F8A                  os9    I$Write
0075 35B6                    puls   d,x,y,pc


          ******************************************
          *
          *     cursor off
          *

0077                offcurs  equ    *
0077 3436                    pshs   d,x,y
0079 308DFFA1                leax   curof,pcr
007D 108E0009                ldy    #curof-curon
0081 8601                    lda    #STDOUT
0083 103F8A                  os9    I$Write
0086 35B6                    puls   d,x,y,pc


          ******************************************
          *
          *     put gcursor sub-command
          *
          *     in  X: buffer
          *         U: flame pointer
          *

0088                putcom   equ    *
0088 3476                    pshs   d,x,y,u
008A CC140F                  ldd    #$140F
008D ED84                    std    ,x
008F EC42                    ldd    2,u           get parameter count
0091 830002                  subd   #2
```

```
0094 ED42                std   2,u
0096 58                  aslb
0097 CB03                addb  #3
0099 E702                stb   2,x        return byte count
009B C609                ldb   #9
009D E703                stb   3,x        send byte count
009F 4F                  clra
00A0 5F                  clrb
00A1 ED04                std   4,x
00A3 CC1F07              ldd   #$1F07     gcursor sub-command + color
00A6 ED06                std   6,x
00A8 EC42                ldd   2,u
00AA 57                  asrb
00AB ED42                std   2,u
00AD E708                stb   8,x

00AF ECD804              ldd   [4,u]
00B2 ED09                std   9,x
00B4 ECD808              ldd   [8,u]
00B7 ED0B                std   11,x

00B9 8603                lda   #3
00BB A70D                sta   13,x

00BD 8601                lda   #STDOUT
00BF 108E000E            ldy   #14
00C3 103F8A              os9   I$Write
00C6 35F6                puls  d,x,y,u,pc



     ****************************************
     *
     *    get gcursor sub-command data
     *
     *    in  X: buffer
     *        U: flame pointer
     *

00C8            getcom   equ   *
00C8 3476                pshs  d,x,y,u
00CA E602                ldb   2,x
00CC CB05                addb  #4+1
00CE 1D                  sex
00CF 1F02                tfr   d,y
00D1 8600                lda   #STDIN
00D3 103F89              os9   I$Read
00D6 35F6                puls  d,x,y,u,pc



     ****************************************
     *
     *    set data
     *
     *    in  X: buffer
     *        U: flame pointer
     *

00D8            setdat   equ   *
00D8 3476                pshs  d,x,y,u
00DA 3007                leax  4+3,x      read index
00DC 314C                leay  12,u       parameter index
00DE            loop     equ   *
00DE EC81                ldd   ,x++
00E0 EDB1                std   [,y++]
00E2 ECA1                ldd   ,y++       parameter.size
00E4 10830002            cmpd  #2
00E8 2611                bne   seterr
00EA EC81                ldd   ,x++
00EC EDB1                std   [,y++]
00EE ECA1                ldd   ,y++
00F0 10830002            cmpd  #2
00F4 2605                bne   seterr
00F6 6A43                dec   3,u
00F8 26E4                bne   loop
00FA 5F                  clrb
```

```
00FB 35F6       seterr   puls  d,x,y,u,pc


00FD E900EA              emod
0100            Size     equ   *
                         end
```

## 3－3－2　circle

circleは円を描くサブルーチンモジュールです。

使用法はF－BASICに準じています。

RUN　circle（［パス］,x,y　,半径　［,　カラー［,　比率
［,開始位置　［,　FNフラグ［,　機能］］］］］）

パス番号はBYTE型の変数でなければならず、他の引数はすべてINTEGER型の変数または定数でなければなりません。

FNフラグは、円の中を塗りつぶすかどうかのフラグで、次の値をとります。

0：　Non-Full
1：　Full　　　（塗りつぶす）

機能は次のうちのいずれかです。

0：　pset
1：　preset
2：　or
3：　and
4：　xor

```
            ****************************************
            *
            *  Basic09 Circle Procedure
            *
```

```
     *    usage: RUN circle( [path#,] x, y, r
     *                 [,color [,ratio [,begin [,end [,FN [,function ]]]
     *
     *          FN      := 0 :N
     *                     1 :F
     *          function:= 0 :pset
     *                     1 :preset
     *                     2 :or
     *                     3 :and
     *                     4 :xor
     *

                         ifp1
                         endc

0000 87CD03EE            mod   Size,Name,Sbrtn+Objct,ReEnt+1,Entry,Mem
000D 43697263   Name     fcs   /Circle/
003C            Mem      equ   60

0015            CSData   equ   *+2        INT(COS(a*PI/64)*$10000)+1 , I
0013 00000001            fdb   0,$0001    a=0
0017 FEC51919            fdb   $FEC5,$1919   1
001B FB1631F2            fdb   $FB16,$31F2   2
001F F4FB4A51            fdb   $F4FB,$4A51   3
0023 EC8461F9            fdb   $EC84,$61F9   4
0027 E1C778AE            fdb   $E1C7,$78AE   5
002B D4DC8E3B            fdb   $D4DC,$8E3B   6
002F C5E5A269            fdb   $C5E5,$A269   7
0033 B506B506            fdb   $B506,$B506   8
FFC4            D$Line   equ   -Mem
0037            L$Data   equ   *
0037 14                  fcb   $14        GID
0038 0E                  fcb   $0E        subout
0039 00                  fcb   $00        return byte count =0
003A 0E                  fcb   $0E        command byte count =$E
003B 00                  fcb   $00        don't care
003C 00                  fcb   $00
003D 15                  fcb   $15        'line' command code
FFCB            C$Code   equ   *-L$Data-Mem
003E 07                  fcb   $07        color code :white
FFCC            F$Code   equ   *-L$Data-Mem
003F 00                  fcb   $00        function code :pset
FFCD            X1       equ   *-L$Data-Mem
0040 0000                fdb   $0000      x1 (dummy)
FFCF            Y1       equ   *-L$Data-Mem
0042 0000                fdb   $0000      y1 (dummy)
FFD1            X2       equ   *-L$Data-Mem
0044 0000                fdb   $0000      x2 (dummy)
FFD3            Y2       equ   *-L$Data-Mem
FFD4            Y2.lo    equ   Y2+1
0046 0000                fdb   $0000      y2 (dummy)
0048 00                  fcb   $00        box flag :straight line
0049 03                  fcb   $03        end code
     *
FFD7            Pass     equ   19-Mem     output pass#
FFD8            FNFlag   equ   20-Mem     F or N flag
FFD9            Ratio    equ   21-Mem     y/x ratio (*$100)
FFDB            O.Pntr   equ   23-Mem     output pointer
FFDC            EndPos   equ   24-Mem     end position
FFDE            Reg1     equ   26-Mem     all-purpose register
FFE0            Reg2     equ   28-Mem
FFE2            Reg3     equ   30-Mem
FFE4            Reg4     equ   32-Mem
FFE5            Reg4.lo  equ   Reg4+1
FFE6            Reg5     equ   34-Mem
FFE8            Reg6     equ   36-Mem
FFEA            Reg7     equ   38-Mem
FFF2            Reg8     equ   46-Mem
FFF4            Reg9     equ   48-Mem
FFF5            Reg9.lo  equ   Reg9+1



004A            Entry    equ   *
004A 33E4                leau  ,s         copy stack ptr
004C 32E8C4              leas  -Mem,s
004F 308CE5              leax  <L$Data,pcr
```

```
00083   0052 31C8C4               leay  <D$Line,u
00084   0055 C613                 ldb   #4+14+1
00085   0057 170382               lbsr  Transmit
00086   005A 8601                 lda   #1
00087   005C A7C8D7               sta   <Pass,u     initial output pass = 1
00088   005F 6FC8D8               clr   <FNFlag,u
00089   0062 8E0073               ldx   #$0073      =0.4495*$100 initial ratio
00090   0065 AFC8D9               stx   <Ratio,u
00091   0068 C601                 ldb   #1
00092   006A E7C8DB               stb   <O.Pntr,u
00093   006D 4F                   clra
00094   006E 5F                   clrb
00095   006F EDC8DC               std   <EndPos,u
00096   0072 EC46                 ldd   6,u         first param byte count
00097   0074 10830002             cmpd  #2
00098   0078 2731                 beq   Circ1
00099   007A 10830001             cmpd  #1
00100   007E 10260363             lbne  ParamErr
00101   0082 A6D804               lda   [4,u]
00102   0085 A7C8D7               sta   <Pass,u
00103   0088 EC42                 ldd   2,u         decrement param count
00104   008A 830001               subd  #1
00105   008D ED42                 std   2,u
00106   008F C103                 cmpb  #3
00107   0091 102D0350             lblt  ParamErr
00108   0095 C109                 cmpb  #9
00109   0097 102E034A             lbgt  ParamErr
00110   009B 5F                   clrb              slide param info.
00111   009C 3144                 leay  4,u
00112   009E AE24         Circ0   ldx   4,y
00113   00A0 AFA1                 stx   ,y++
00114   00A2 AE24                 ldx   4,y
00115   00A4 AFA1                 stx   ,y++
00116   00A6 5C                   incb
00117   00A7 E143                 cmpb  3,u
00118   00A9 26F3                 bne   Circ0
00119   00AB EC42         Circ1   ldd   2,u         get param count
00120   00AD 10830009             cmpd  #9          check param count
00121   00B1 102E0330             lbgt  ParamErr
00122   00B5 10830003             cmpd  #3
00123   00B9 102D0328             lblt  ParamErr
00124        *
00125   00BD 4F                   clra
00126   00BE 5F                   clrb
00127   00BF 3406                 pshs  d
00128        *  Parameter x
00129   00C1 EC46                 ldd   6,u         check integer?
00130   00C3 10830002             cmpd  #2
00131   00C7 1026031A             lbne  ParamErr
00132   00CB 6A43                 dec   3,u
00133        *  Parameter y
00134   00CD EC4A                 ldd   10,u        integer?
00135   00CF 10830002             cmpd  #2
00136   00D3 1026030E             lbne  ParamErr
00137   00D7 6A43                 dec   3,u
00138        *  Parameter r
00139   00D9 EC4E                 ldd   14,u        integer?
00140   00DB 10830002             cmpd  #2
00141   00DF 10260302             lbne  ParamErr
00142   00E3 6A43                 dec   3,u
00143   00E5 102700DD             lbeq  Circ2
00144        *  Parameter color
00145   00E9 ECC812               ldd   18,u        integer?
00146   00EC 10830002             cmpd  #2
00147   00F0 102602F1             lbne  ParamErr
00148   00F4 ECD810               ldd   [16,u]      get color#
00149   00F7 102B02EA             lbmi  ParamErr
00150   00FB 10830007             cmpd  #7
00151   00FF 102E02E2             lbgt  ParamErr
00152   0103 E7C8CB               stb   <C$Code,u
00153   0106 6A43                 dec   3,u
00154   0108 102700BA             lbeq  Circ2
00155        *  Parameter ratio
00156   010C ECC816               ldd   22,u        real?
00157   010F 10830005             cmpd  #5
00158   0113 102602CE             lbne  ParamErr
00159   0117 10AEC814             ldy   20,u
```

```
011B A624                lda   4,y
011D 8501                bita  #1
011F 102602C2            lbne  ParamErr
0123 A6A4                lda   ,y
0125 8B08                adda  #8
0127 AE21                ldx   1,y
0129 170296              lbsr  CInt
012C AFC8D9              stx   <Ratio,u      Ratio=int(ratio*$100)
012F 6A43                dec   3,u
0131 10270091            lbeq  Circ2
     *  Parameter begin
0135 ECC81A              ldd   26,u          real?
0138 10830005            cmpd  #5
013C 102602A5            lbne  ParamErr
0140 10AEC818            ldy   24,u
0144 A6A4                lda   ,y
0146 8B06                adda  #6
0148 AE21                ldx   1,y
014A 170275              lbsr  CInt
014D 1F10                tfr   x,d
014F C43F                andb  #%00111111
0151 1F98                tfr   b,a
0153 C407                andb  #%00000111
0155 44                  lsra
0156 44                  lsra
0157 44                  lsra
0158 EDE4                std   ,s
015A 6A43                dec   3,u
015C 2768                beq   Circ2
     *  Parameter end
015E ECC81E              ldd   30,u          real?
0161 10830005            cmpd  #5
0165 1026027C            lbne  ParamErr
0169 10AEC81C            ldy   28,u
016D A6A4                lda   ,y
016F 8B06                adda. #6
0171 AE21                ldx   1,y
0173 17024C              lbsr  CInt
0176 1F10                tfr   x,d
0178 C43F                andb  #%00111111
017A 1F98                tfr   b,a
017C C407                andb  #%00000111
017E 44                  lsra
017F 44                  lsra
0180 44                  lsra
0181 EDC8DC              std   <EndPos,u
0184 6A43                dec   3,u
0186 273E                beq   Circ2
     *  Parameter FN
0188 ECC822              ldd   34,u          integer?
018B 10830002            cmpd  #2
018F 10260252            lbne  ParamErr
0193 ECD820              ldd   [32,u]
0196 270D                beq   FN1           0:N
0198 10830001            cmpd  #1            1:F
019C 10260245            lbne  ParamErr
01A0 C6FF                ldb   #$FF
01A2 E7C8D8              stb   <FNFlag,u
01A5 6A43       FN1      dec   3,u
01A7 271D                beq   Circ2
     *  Parameter function
01A9 ECC826              ldd   38,u          integer?
01AC 10830002            cmpd  #2
01B0 10260231            lbne  ParamErr
01B4 ECD824              ldd   [36,u]        get function code
01B7 102B022A            lbmi  ParamErr
01BB 10830004            cmpd  #4
01BF 102E0222            lbgt  ParamErr
01C3 E7C8CC              stb   <F$Code,u
     *
01C6            Circ2    equ   *
01C6 3506                puls  d
     *
01C8            Circloop equ   *
01C8 3406                pshs  d
01CA AEC8D1              ldx   <X2,u
01CD AFC8CD              stx   <X1,u
```

```
00237    01D0 AEC8D3              ldx   <Y2,u
00238    01D3 AFC8CF              stx   <Y1,u
00239    01D6 8401                anda  #%001
00240    01D8 2703                beq   Circ3
00241    01DA 50                  negb
00242    01DB CB08                addb  #8             --->3 bits
00243    01DD 58         Circ3    aslb                 b*4--->b
00244    01DE 58                  aslb
00245    01DF 318DFE32            leay  CSData,pcr
00246    01E3 31A5                leay  b,y
00247    01E5 3420                pshs  y
00248    01E7 1700A6              lbsr  Calc
00249    01EA 3520                puls  y
00250    01EC 313E                leay  -2,y
00251    01EE 3410                pshs  x
00252    01F0 17009D              lbsr  Calc
00253    01F3 3520                puls  y
00254    01F5 A6E4                lda   ,s
00255    01F7 8403                anda  #%011
00256    01F9 2706                beq   Circ4
00257    01FB 8103                cmpa  #%011
00258    01FD 2702                beq   Circ4
00259    01FF 1E12                exg   x,y
00260    0201 AFC8D1     Circ4    stx   <X2,u
00261    0204 1F21                tfr   y,x
00262    0206 ECC8D9              ldd   <Ratio,u
00263    0209 170093              lbsr  Mul16
00264    020C 1E20                exg   y,d
00265    020E 4D                  tsta
00266    020F 102601D2            lbne  ParamErr
00267    0213 E7C8D3              stb   <Y2,u
00268    0216 1F20                tfr   y,d
00269    0218 A7C8D4              sta   <Y2.lo,u
00270    021B A6E4                lda   ,s
00271    021D 8102                cmpa  #%010
00272    021F 2510                blo   Circ5
00273    0221 8106                cmpa  #%110
00274    0223 240C                bhs   Circ5
00275    0225 ECD804              ldd   [4,u]          get center x positin
00276    0228 A3C8D1              subd  <X2,u
00277    022B 2415                bhs   Circ7
00278    022D 4F                  clra
00279    022E 5F                  clrb
00280    022F 2011                bra   Circ7
00281    0231 ECD804     Circ5    ldd   [4,u]          center x
00282    0234 E3C8D1              addd  <X2,u
00283    0237 2506                bcs   Circ6
00284    0239 1083027F            cmpd  #639           x limit
00285    023D 2503                blo   Circ7
00286    023F CC027F     Circ6    ldd   #639
00287    0242 EDC8D1     Circ7    std   <X2,u
00288    0245 A6E4                lda   ,s
00289    0247 8104                cmpa  #%100
00290    0249 250D                blo   Circ8
00291    024B ECD808              ldd   [8,u]          get center y
00292    024E A3C8D3              subd  <Y2,u
00293    0251 2416                bcc   Circ10
00294    0253 CC0000              ldd   #0
00295    0256 2011                bra   Circ10
00296    0258 ECD808     Circ8    ldd   [8,u]          center y
00297    025B E3C8D3              addd  <Y2,u
00298    025E 2506                bcs   Circ9
00299    0260 108300C7            cmpd  #199           y limit
00300    0264 2503                blo   Circ10
00301    0266 CC00C7     Circ9    ldd   #199
00302    0269 EDC8D3     Circ10   std   <Y2,u
00303    026C 6DC8DB              tst   <O.Pntr,u
00304    026F 2602                bne   Circ11
00305    0271 8D5B                bsr   S$Line
00306    0273 3506       Circ11   puls  a,b
00307    0275 64C8DB              lsr   <O.Pntr,u
00308    0278 2506                bcs   Circ12
00309    027A 10A3C8DC            cmpd  <EndPos,u
00310    027E 270C                beq   Return
00311    0280 5C         Circ12   incb
00312    0281 C108                cmpb  #%00001000
00313    0283 2604                bne   Circ13
```

```
0285 4C                   inca
0286 5F                   clrb
0287 8407                 anda  #%00000111
0289 16FF3C     Circ13    lbra  Circloop
     *
     *  Return to Basic
     *
028C            Return    equ   *
028C 32C4                 leas  ,u
028E 4F                   clra                cc reset
028F 39                   rts
     *
     *  X,Y:radius*((Y)-1)'s upper 16 bits
     *
0290            Calc      equ   *
0290 AED80C               ldx   [12,u]        get radius
0293 ECA4                 ldd   ,y            get sin/cos data
0295 2707                 beq   Calc1
0297 830001               subd  #1
029A 8D03                 bsr   Mul16
029C 1F21                 tfr   y,x
029E 39         Calc1     rts
     *
     *  D*X--->Y:upper 16 bits ; d:lower 16 bits
     *
029F            Mul16     equ   *
029F 3410                 pshs  x             dummy
02A1 3436                 pshs  d,x,y
02A3 6F64                 clr   4,s
02A5 A663                 lda   3,s
02A7 3D                   mul
02A8 ED66                 std   6,s
02AA EC61                 ldd   1,s
02AC 3D                   mul
02AD EB66                 addb  6,s
02AF 8900                 adca  #0
02B1 ED65                 std   5,s
02B3 E6E4                 ldb   ,s
02B5 A663                 lda   3,s
02B7 3D                   mul
02B8 E365                 addd  5,s
02BA ED65                 std   5,s
02BC 2402                 bhs   Mul.1
02BE 6C64                 inc   4,s
02C0 A6E4       Mul.1     lda   ,s
02C2 E662                 ldb   2,s
02C4 3D                   mul
02C5 E364                 addd  4,s
02C7 ED64                 std   4,s
02C9 3536                 puls  d,x,y
02CB 3506                 puls  d
02CD 39                   rts
     *
     *  Draw Line (main)
     *
02CE            S$Line    equ   *
02CE 6DC8D8               tst   <FNFlag,u
02D1 2603                 bne   Ln1
02D3 1600D6               lbra  Line
     *
02D6 31C4       Ln1       leay  ,u
02D8 8E0001               ldx   #1
02DB CEFFFF               ldu   #-1
02DE ECA8D1               ldd   <X2,y
02E1 A3A8CD               subd  <X1,y
02E4 2B05                 bmi   Ln2
02E6 AFA8E2               stx   <Reg3,y
02E9 2009                 bra   Ln3
02EB EFA8E2     Ln2       stu   <Reg3,y
02EE ECA8CD               ldd   <X1,y
02F1 A3A8D1               subd  <X2,y
02F4 C30001     Ln3       addd  #1
02F7 EDA8DE               std   <Reg1,y
02FA EDA8E8               std   <Reg6,y
02FD ECA8D3               ldd   <Y2,y
0300 A3A8CF               subd  <Y1,y
0303 2B05                 bmi   Ln4
```

```
0305 AFA8E4                    stx    <Reg4,y
0308 2009                      bra    Ln5
030A EFA8E4          Ln4       stu    <Reg4,y
030D ECA8CF                    ldd    <Y1,y
0310 A3A8D3                    subd   <Y2,y
0313 C30001          Ln5       addd   #1
0316 EDA8E0                    std    <Reg2,y
0319 4F                        clra
031A 5F                        clrb
031B EDA8E6                    std    <Reg5,y
031E ECA8CD                    ldd    <X1,y
0321 EDA8F2                    std    <Reg8,y
0324 ECA8CF                    ldd    <Y1,y
0327 EDA8F4                    std    <Reg9,y
032A 8D4D            Ln6       bsr    Ln10
032C ECA8E6                    ldd    <Reg5,y
032F E3A8E0                    addd   <Reg2,y
0332 EDA8E6                    std    <Reg5,y
0335 A3A8DE          Ln7       subd   <Reg1,y
0338 231B                      bls    Ln8
033A EDA8E6                    std    <Reg5,y
033D E6A8F5                    ldb    <Reg9.lo,y
0340 EBA8E5                    addb   <Reg4.lo,y
0343 E7A8F5                    stb    <Reg9.lo,y
0346 ECA8E6                    ldd    <Reg5,y
0349 A3A8DE                    subd   <Reg1,y
034C 2515                      blo    Ln9
034E 8D29                      bsr    Ln10
0350 ECA8E6                    ldd    <Reg5,y
0353 20E0                      bra    Ln7
0355 260C            Ln8       bne    Ln9
0357 EDA8E6                    std    <Reg5,y
035A E6A8F5                    ldb    <Reg9.lo,y
035D EBA8E5                    addb   <Reg4.lo,y
0360 E7A8F5                    stb    <Reg9.lo,y
0363 ECA8F2          Ln9       ldd    <Reg8,y
0366 E3A8E2                    addd   <Reg3,y
0369 EDA8F2                    std    <Reg8,y
036C AEA8E8                    ldx    <Reg6,y
036F 301F                      leax   -1,x
0371 AFA8E8                    stx    <Reg6,y
0374 26B4                      bne    Ln6
0376 33A4                      leau   ,y
0378 39                        rts
     *
0379                 Ln10      equ    *
0379 33A4                      leau   ,y
037B C608                      ldb    #8
037D 30C8CD                    leax   <X1,u
0380 31C8EA                    leay   <Reg7,u
0383 8D57                      bsr    Transmit
0385 ECD804                    ldd    [4,u]
0388 EDC8CD                    std    <X1,u
038B ECD808                    ldd    [8,u]
038E EDC8CF                    std    <Y1,u
0391 ECC8F2                    ldd    <Reg8,u
0394 EDC8D1                    std    <X2,u
0397 ECC8F4                    ldd    <Reg9,u
039A EDC8D3                    std    <Y2,u
039D 8D0D                      bsr    Line
039F C608                      ldb    #8
03A1 30C8EA                    leax   <Reg7,u
03A4 31C8CD                    leay   <X1,u
03A7 8D33                      bsr    Transmit
03A9 31C4                      leay   ,u
03AB 39                        rts
     *
     *  Draw line (sub)
     *
03AC                 Line      equ    *
03AC 3432                      pshs   a,x,y
03AE 30C8C4                    leax   <D$Line,u
03B1 108E0013                  ldy    #4+14+1
03B5 A6C8D7                    lda    <Pass,u
03B8 103F8A                    os9    I$Write
03BB 2502                      bcs    IOErr
03BD 35B2                      puls   a,x,y,pc
```

```
03BF 32C4       IOErr    leas  ,u
03C1 39                  rts                  return to Basic09 System
     *
     *  Real--->Integer
     *
03C2            CInt     equ   *
03C2 3412                pshs  a,x
03C4 4D                  tsta
03C5 2A06                bpl   CInt1
03C7 6F61                clr   1,s
03C9 6F62                clr   2,s
03CB 200D                bra   CInt2
03CD 8010       CInt1    suba  #16
03CF 2E14                bgt   ParamErr
03D1 2707                beq   CInt2
03D3 6461       CIntLoop lsr   1,s
03D5 6662                ror   2,s
03D7 4C                  inca
03D8 26F9                bne   CIntLoop
03DA 3592       CInt2    puls  a,x,pc
     *
     *   Block Transmission
     *
03DC            Transmit equ   *
03DC 5A                  decb
03DD A685       T$loop   lda   b,x
03DF A7A5                sta   b,y
03E1 5A                  decb
03E2 2CF9                bge   T$loop
03E4 39                  rts
     *
     *  Parameter Error
     *
03E5            ParamErr equ   *
03E5 32C4                leas  ,u
03E7 C638                ldb   #56
03E9 43                  coma
03EA 39                  rts


03EB B4B4C3              emod
03EE            Size     equ   *
                         end
```

## 3—3—3　sound

soundはF—BASICのSOUND文に相当するサブルーチンモジュールです。使用法はF—BASICに準じています。

```
RUN　sound（PSGレジスタ番号，データ）
```

引数は２つともINTEGER型の変数または定数でなければなりません。またレジスタ番号およびデータの形式の詳細は、F—BASICの文法書を参照して下さい。

```
     ****************************************
     *
     *  Basic09 Sound Procedure
     *
     *    usage: RUN sound( PSG-Register#, Data )
     *

                            ifp1
                            endc

0000 87CD005E               mod     Size,Name,Sbrtn+Objct,ReEnt+1,Entry,Mem
000D 536F756E     Name      fcs     /Sound/
0020              Mem       equ     32

FD0D              PSG.Reg   equ     $FD0D         PSG Register addr.


0012              Entry     equ     *
0012 33E4                   leau    ,s            copy stack ptr
0014 EC42                   ldd     2,u           parameter check
0016 10830002               cmpd    #2
001A 2639                   bne     Error
001C EC46                   ldd     6,u
001E 10830002               cmpd    #2
0022 2631                   bne     Error
0024 EC4A                   ldd     10,u
0026 10830002               cmpd    #2
002A 2629                   bne     Error
002C ECD804                 ldd     [4,u]         get first parameter
002F 2B24                   bmi     Error         if minus then error
0031 1083000D               cmpd    #13           if >13 then error
0035 221E                   bhi     Error
0037 3404                   pshs    b
0039 ECD808                 ldd     [8,u]         get second parameter
003C 8100                   cmpa    #0            if >$FF then error
003E 2615                   bne     Error
0040 3502                   puls    a
0042 8EFD0D                 ldx     #PSG.Reg
0045 A701                   sta     1,x           store reg#
0047 8603                   lda     #3            reg-appointment command
0049 A784                   sta     ,x
004B 6F84                   clr     ,x            clear command-reg
004D E701                   stb     1,x           store data
004F 4A                     deca                  write command
0050 A784                   sta     ,x
```

```
00047   0052 6F84               clr    ,x          clear command-reg
00048   0054 39                 rts
00049
00050   0055 C638       Error   ldb    #$38
00051   0057 43                 coma
00052   0058 32C4               leas   ,u
00053   005A 39                 rts
00054
00055
00056   005B 8AB8DE             emod
00057   005E            Size    equ    *
00058                           end
```

## 3―3―4　geditor

このプログラムは対話形式で絵を作成するプログラムです。

実行するには、Basic09からロードしてＲＵＮするか、またはＰＡＣＫしてあればShell コマンドラインからコマンドとして起動します。すると、

1）Edit　2）Load　3）Save　CR）End

とメニューを表示し、入力を促してくるので、１文字入力します。

それでは、各モードを説明します。

1）Edit

このモードに入ると、

1）Connect　2）Box　3）Circle　4）Paint　5）Pset　6）Delete
7）Clear　CR）End

と表示されますので、それぞれのキーを入力してください。

各メニューの内容は次の通りです。

① Connect

連続した直線を描くときに使います。

まず、グラフィックカーソルが出るので、カーソルキーで動かして始点を決め、次に終点を決めます。次に、色を決めます。

Color　0) Black　1　2　3　4　5　6　7　CR) No　change
　　　　　　　　└──────┘そのカラーコード色で表示

と表示されますので、好きな色を入力します。このときCRキーを入力すると、前回と同じになります。次に機能を決めます。

0) Pset　1) Preset　2) Or　3) And　4) Xor
　　　　　　　　　　　　　5) Not　CR) No　change

となりますので同様に入力します。さらに、

CR) End　ELSE) Continue

と表示されます。次に移りたければCRキーを、続けたければそれ以外のキーを入力します。続けるときには、前回の終点が今度の始点になりますので、座標は、終点のみ決めます。

② Box

長方形および長方形の塗りつぶしを描きます。

グラフィックカーソルで対角線の両端の位置を決め、色、機能を決めます。次に、

0) No　fill　1) Fill　CR) No　change

と塗りつぶすかどうか聞いてきますので、0、1、またはCRキーを入力します。そして、続けるときはCRキー以外押しますが、Connectのときと違って、対角線の両端を決める必要があります。(以下同様)

③ Circle

円および円の塗りつぶしを描きます。

グラフィックカーソルで円の中心と円周上の一点を決め、色、機能、塗りつぶすかどうかを決めます。

④ Paint

指定された領域を塗りつぶします

グラフィックカーソルで領域を決めたあと、境界色、塗りつぶす色の順に

決めます。このとき、画面はクリアされ、絵は最初から描き直されますが、これは、画面の下２桁がコマンド表示に使われているため、指定された領域の境界色がこの部分にあった場合、コマンド表示によって消されてしまい余計な所まで塗りつぶされてしまう可能性があるためです。なお、現在の所、境界色は１色（塗る色と合わせて２色）しか指定できません。

⑤ Pset

点を打ちます。

グラフィックカーソルで位置を決め、機能を決めます。機能を 1) Preset にしたとき以外は色を指定します。

⑥ Delete

直前のデータを削除します。

削除してよいかどうか聞いてきますので、よければＣＲキー、そうでなければそれ以外のキーを押してください。絵は最初から描き直されます。

⑦ Clear

データをすべて削除して、画面をクリアします。

クリアしてもよいかどうか聞いてきますので、よければCRキーを押してください。画面はクリアされます。

データは３００個まで入りますが、増やしたい時は、配列 dat の第一添字と max の値を変えて下さい。ただし、

dat の第一添字＝max＋１

です。データがいっぱいになると、"Date　area　full"のメッセージが出てメインメニューに戻ります。

２）Load

データファイルから読み込みます。

ファイル名の入力を促してきますので読み込むファイル名を入力してください。

存在しないファイル名を入力するとメインメニューに戻ります。

データはファイル上では次のようなグラフィックデータの形になっています。

```
line   14 0e 00 10 00 00 15 □ □ □□ □□ □□ □□ □ xx xx 03
                            co fu xs  ys  xe  ye fill


paint 14 0e 00 0a 00 00 18 □□ □□ □ 01 □ 03
                           xs  ys co   bco


pset  14 0e 00 0a 00 00 17 01 □□ □□ □ □ 03
                              xs  ys co fu
```

circle は line の集合として表されます。ただし、

```
14 0e 00 0e 00 00 15 □ □ □□ □□ □□ □□ □ 03
```

という形になり、普通の line とは形式が少し違っています。(２バイト分データが少ない。)

このデータを配列 dat に変換して画面に描きます。詳しいことは、各ＰＲＯＣＥＤＵＲＥの説明の　４）　load_dataの所を見てください。

３）Save

データをグラフィックのままの形でセーブします。

ファイル名を入力してください。このとき、既に存在しているファイル名や、ＣＲキーのみ入力すると、メインメニューに戻ります。

４）End

ＯＳ－９に戻ります。

データをセーブしたかどうか聞いてくるので、セーブしてあるのならＣＲキー、そうでなければそれ以外のキーを入力します。

各ＰＲＯＣＥＤＵＲＥの説明

○ geditor

メインルーチンです。ここからEdit、Load、Save、Endのルーチンに分岐します。

○ g_edit

Editルーチンです。グラフィック各コマンドの処理ルーチンに分岐します。

○ save_data

データを内部形式にしてセーブします。

○ load_data

グラフィックデータの内部表現を配列のデータに直して、画面に出力します。各グラフィックの最初の３バイトは座標、色などの情報をもっていないので読み飛ばします。４バイト目は$10ならline、$0Aならpaintまたはpset、それ以外ならcircleとなり、それぞれ１０００、２０００、３０００に分岐します。paintとpsetの区別は、７バイト目が$17ならpset、$18ならpaintであり、それぞれ２１００、２２００に分岐します。

line、pset、paintの内部表現は前出の通りですが、circleはlineの集合として表されるため、特別の処理が必要になります。

ａ）塗りつぶしをしない場合

右端の点から右まわりに短い直線によって描かれています。最初の線の始点ｙ座標から中心のｙ座標が求まり、次にこの始点のｘ座標xsと、終点のｙ座標から中心のy座標に等しくなる直線の終点のｘ座標ｘｅから半径ｒが、

$$r = (xs - xe) / 2$$

で中心のｘ座標が、

$$x = xs - r$$

で求められます。

ｂ）塗りつぶす場合

右端の点から右まわりに、中心から直線を引くことによって描かれていま

す。最初の直線の始点が中心の座標となり、始点と終点のx座標の差が半径rになります。

塗りつぶすかどうかの判別は、最初の直線を調べ、始点と終点のy座標が等しいかどうかで判別します。すなわち、等しければ塗りつぶし、そうでなければ塗りつぶさないということがわかります。

○ draw_connect

Connectの処理ルーチンです。

xs、ys ………始点のx、y座標
xe、ye ………終点のx、y座標
fu ………機能
co ………色

○ draw_box

Boxの処理ルーチンです。

fill ………0：輪郭　1：塗りつぶし
他はConnectと同じです。

○ draw_circle

Circleの処理ルーチンです。

xs、ys ………円の中心のx、y座標
xe、ye ………円周上の点のx、y座標
r ………半径

○ draw_paint

Paintの処理ルーチンです。

xs、ys ………Paintの開始点のx、y座標
bco ………境界色

○ draw_pset

Psetの処理ルーチンです。

xs、ys ………点のx、y座標

○ Delete_data

Deleteの処理ルーチンです。numを1だけ減らしています。

○ Clear

Clearの処理ルーチンです。numを0にします。

○ input_color

カラーコードの入力を処理します。境界色の入力にも使っています。

○ input_fill

塗りつぶすかどうか（fill）の入力を処理します。BoxおよびCircle用です。

○ input_function

機能コードの入力を処理します。

○ loc023

テキスト画面の下2行にあたる部分をテキスト、グラフィックともに消してカーソルを（0，23）の位置にもっていきます。

○ draw

絵を最初から描き直します。パス番号pathのファイルまたは画面に出力します。Save、Load、Paint、Deleteで使います。

○ cls

テキスト、グラフィック画面両方をクリアします。

プログラム中で共通に使用している変数を説明します。

num　………グラフィックデータの番号。0から始まります。

max　………numの最大値。max＋1個のデータが入力可能です。

dat (300, 8)　………グラフィックデータが入ります。第1添字はグラフィックの番号を、第2添字はグラフィックコマンドによって次のように変わります。

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Connect | 1 | xs | ys | xe | ye | fu | co | 0 |
| Box | 1 | xs | ys | xe | ye | fu | co | fill |
| Circle | 2 | xs | ys | r | co | fill | fu | — |
| Paint | 3 | xs | ys | co | bco | — | — | — |
| Pset | 4 | xs | ys | co | fu | — | — | — |

ch　………1文字入力用の文字変数

（FM－7, 8用）

```
PROCEDURE geditor
(*    definitions    *)
base 0
dim num, cmd, dat(300,8): integer
dim path: byte
dim ch: string[1]

    num = 0
    (*    main routine    *)
    run cls
    loop
        run loc023
        print "1)edit 2)load 3)save cr)end";
        get #0, ch
        if ch = hr$(13) then
            run loc023
            print "data already saved cr)ok else)no";
            get #0, ch
            if ch = chr$(13) then
                end
            else
                run save_data( num, dat )
                end
            endif
        else
            if ch > "0" or ch < "4" then
                cmd = val( ch )
                on cmd gosub 10, 20, 30
            endif
        endif
    endloop

    10 run g_edit(num,dat)      ¥ return
    20 run load_data(num,dat)  ¥ return
    30 run save_data(num,dat)  ¥ return
end


PROCEDURE g_edit
```

```
(*    edit routine    *)
base 0
param num: integer
param dat(300,8): integer
dim max, xs, ys, cmd: integer
dim ch: string[1]

    max = 299
    loop
        run loc023
        if num > max then
            run loc023
            print "data area full"
            end
        endif
        run loc023
        print"1)connect 2)box 3)circle 4)paint 5)pset 6)delete 7)clear cr)end";
        get #0, ch
        if ch = chr$(13) then
            end
        else
            if ch > "0" and ch < "8" then
                cmd = val(ch)
                on cmd gosub 10, 20, 30, 40, 50, 60, 70
            endif
        endif
    endloop

10  run draw_connect( num, dat ) ¥ return
20  run draw_box( num, dat )     ¥ return
30  run draw_circle( num, dat )  ¥ return
40  run draw_paint( num, dat )   ¥ return
50  run draw_pset( num, dat )    ¥ return
60  run delete_data( num, dat )  ¥ return
70  run clear( num, dat )        ¥ return
end


PROCEDURE save_data
(*    save routine    *)
base 0
param num: integer
param dat(300,8): integer
dim i, j: integer
dim path: byte

    on error goto 10
    run loc023
    input "file name ", f$
    if f$ = "" then
        end
    endif
    create #path, f$: write
    run draw( path, num, dat )
    close #path
    end

10
    if err = 218 then
        run loc023
        print "file already exists";
    else if err=215 then
        run loc023
        print "bad path name";
    endif ¥ endif
end


ROCEDURE draw_connect
(*    connect routine    *)
base 0
param num: integer
param dat(300,8): integer
dim max, xs, ys, xe, ye, co, fu: integer
dim ch: string[1]
```

```
    max = 299
    xs = 320
    ys = 100
    co = 7
    fu = 0
    run gcursor( xs, ys, xs, ys )
    loop
        run pset( xs, ys, 7, 4 )
        run gcursor( xs, ys, xe, ye )
        run loc023
        run input_color( co )
        run input_function( fu )
        run pset( xs, ys, 7, 4 )
        run line( xs, ys, xe, ye, fu, co )
        dat( num, 0 ) = 1
        dat( num, 1 ) = xs
        dat( num, 2 ) = ys
        dat( num, 3 ) = xe
        dat( num, 4 ) = ye
        dat( num, 5 ) = fu
        dat( num, 6 ) = co
        dat( num, 7 ) = 0
        num = num + 1
        if num > max then
            end
        endif
        xs = xe
        ys = ye
        run loc023
        print "cr)end else)continue";
        get #0, ch
        if ch = chr$(13) then
            end
        endif
    endloop
end



PROCEDURE draw_box
(*    box, boxfill routine    *)
base 0
param num: integer
param dat(300,8): integer
dim max, xs, ys, xe, ye, co, fu, fill: integer
dim ch: string[1]

    max = 299
    xs = 320
    ys = 100
    co = 7
    fu = 0
    fill = 0

    loop
        run gcursor( xs, ys, xs, ys )
        run pset( xs, ys, 7, 4 )
        run gcursor( xs, ys, xe, ye )
        run loc023
        run input_color( co )
        run input_function( fu )
        run input_fill( fill )
        run pset( xs, ys, 7, 4 )
        run line( xs, ys, xe, ye, fu, co, fill+1 )
        dat( num, 0 ) = 1
        dat( num, 1 ) = xs
        dat( num, 2 ) = ys
        dat( num, 3 ) = xe
        dat( num, 4 ) = ye
        dat( num, 5 ) = fu
        dat( num, 6 ) = co
        dat( num, 7 ) = fill+1
        num = num + 1
        if num > max then
            return
        endif
        xs = xe
        ys = ye
        run loc023
        print "cr)end else)continue";
        get #0, ch
        if ch = chr$(13) then
            end
        endif
    endloop
end


PROCEDURE draw_circle
(*    circle routine    *)
base 0
param num: integer
param dat(300,8): integer
end


PROCEDURE draw_paint
(*    paint routine    *)
base 0
param num: integer
param dat(300,8): integer
dim max, xs, ys, co, bco: integer
dim path: byte
dim ch: string[1]

    max = 299
    xs = 320
    ys = 100
    co = 7
    bco = 7
    loop
        run gcursor( xs, ys, xs, ys )
        run pset( xs, ys, 7, 4 )
        run loc023
        print "boundary ";
        run input_color( co )
        bco = co
        run loc023
        run input_color( co )
        dat( num, 0 ) = 3
        dat( num, 1 ) = xs
        dat( num, 2 ) = ys
        dat( num, 3 ) = co
        dat( num, 4 ) = bco
        num = num + 1
        run cls
        open #path, "/term": write
        run draw( path, num, dat )
        close #path
        if num > max then
            return
        endif
        run loc023
        print "cr)end else)continue";
        get #0, ch
        if ch = chr$(13) then
            end
        endif
    endloop
end


PROCEDURE draw_pset
(*    pset, preset routine    *)
base 0
param num: integer
param dat(300,8): integer
end
```

```
PROCEDURE delete_data
(*    delete routine    *)
base 0
param num: integer
param dat(300,8): integer
dim path: byte
dim ch: string[1]

    if num = 0 then
        run loc023
        print "data empty"
        end
    endif
    run loc023
    print "delete ok ? cr)ok else)no";
    get #0, ch
    if ch = chr$(13) then
        num = num - 1
        run cls
        open #path, "/term": write
        run draw( path, num, dat )
        close #path
    endif
end


PROCEDURE clear
(*    clear routine    *)
base 0
param num: integer
param dat(300,8): integer
dim ch: string[1]

    run loc023
    print "clear ok ? cr)ok else)no";
    get #0, ch
    if ch = chr$(13) then
        num = 0
        run cls
    endif
end


PROCEDURE input_color
(*    color code input routine    *)
param co: integer
dim i: integer
dim ch: string[1]

    loop
        print "color  0)black ";
        for i = 1 to 7
            run color(i)
            print i; " ";
        next i
        print "cr)no change";
        get #0, ch
        if ch = chr$(13) then
            end
        else
            if ch >= "0" and ch <= "7" then
                co = val(ch)
                end
            endif
            run loc023
        endif
    endloop
end


PROCEDURE input_fill
(*    fill ?    *)
param fill: integer
dim ch: string[1]
```

```
    loop
        run loc023
        print "0)no fill 1)fill cr)no change";
        get #0, ch
        if ch = chr$(13) then
            end
        else if ch >= "0" and ch <= "1" then
            fill = val(ch)
            end
        endif ¥ endif
    endloop
end



PROCEDURE input_function
(*    function code input routine    *)
param fu: integer
dim ch: string[1]

    loop
        run loc023
        print "0)pset 1)preset 2)or 3)and 4)xor 5)not cr)no change";
        get #0, ch
        if ch = chr$(13) then
            end
        else if ch >= "0" and ch <= "5" then
            fu = val(ch)
            end
        endif ¥ endif
    endloop
end



PROCEDURE loc023
(*    locate 0,23    *)

    run line( 0, 184, 639, 199, 0, 0, 2 )
    run locate( 0, 23 )
    print "
    print "
    run locate( 0, 23 )
end



PROCEDURE draw
(*    draw routine    *)
base 0
param path: byte
param num: integer
param dat(300,8): integer
dim i: integer

    for i = 0 to num-1
        if dat(i,0) = 1 then
            run line( path, dat(i,1), dat(i,2), dat(i,3), dat(i,4), dat(i,5),
                                                    dat(i,6), dat(i,7) )
        else if dat(i,0) = 2 then
            run circle( path, dat(i,1), dat(i,2), dat(i,3), dat(i,4), .4495,
                                                    .0,1., dat(i,5), dat(i,6) )
        else if dat(i,0) = 3 then
            run paint( path, dat(i,1), dat(i,2), dat(i,3), dat(i,4) )
        else if dat(i,0) = 4 then
            run pset( path, dat(i,1), dat(i,2), dat(i,3), dat(i,4) )
        endif ¥ endif ¥ endif ¥ endif
    next i
end

PROCEDURE cls
dim b(1): byte

    b(1) = $16
    put #1, b
```

```
end


PROCEDURE load_data
(*   load routine    *)
base 0
param num: integer
param dat(300,8): integer
dim dummy1: string[3]
dim dummy2: string[15]
dim dummy3: string[2]
dim dummy4: string[19]
dim i, j, tempx, tempy, xs, ys, xe, ye: integer
dim path, byt: byte

    on error goto 10
    run loc023
    input "file name ", f$
    if f$ = "" then
        end
    endif
    open #path, f$: read
    i = 0
    while not( eof( #path ) ) do
        get #path, dummy1
        get #path, byt
        if byt = $10 then
            gosub 1000
        else if byt = $0a then
            gosub 2000
        else
            gosub 3000
        endif ¥ endif
        i = i + 1
        get #path, byt
    endwhile
    num = i
    close #path
    run cls
    open #path, "/term": write
    run draw( path, num, dat )
    close #path
    end

10
    if err = 216 then
        run loc023
        print "file not found";
    else if err = 215 then
        run loc023
        print "bad path name";
    endif ¥ endif
    end

1000
    get #path, dummy1
    get #path, byt
    dat(i,6) = byt
    get #path, byt
    dat(i,5) = byt
    for j = 1 to 4
        get #path, dat(i,j)
    next j
    get #path, byt
    dat(i,7) = byt
    dat(i,0) = 1
    get #path, dummy3
    return

2000
    get #path, dummy1
    byt = asc( right$( dummy1, 1 ) )
    if byt = $17 then
        gosub 2100
    else
        gosub 2200
    endif
    return

2100
    get #path, byt
    for j = 1 to 2
        get #path, dat(i,j)
    next j
    get #path, byt
    dat(i,3) = byt
    get #path, byt
    dat(i,4) = byt
    dat(i,0) = 4
    return

2200
    for j = 1 to 2
        get #path, dat(i,j)
    next j
    get #path, byt
    dat(i,3) = byt
    get #path, byt
    get #path, byt
    dat(i,4) = byt
    dat(i,0) = 3
    return

3000
    dat(i,0) = 2
    get #path, dummy1
    get #path, byt
    dat(i,4) = byt
    get #path, byt
    dat(i,6) = byt
    get #path, xs
    get #path, ys
    get #path, xe
    get #path, ye
    if ys = ye then
        dat(i,1) = xs
        dat(i,2) = ys
        dat(i,3) = xe - xs
        dat(i,5) = 1
        tempx = xe - 1
        tempy = ye - 1
        while xe <> tempx or ye <> tempy do
            get #path, dummy2
            get #path, tempx
            get #path, tempy
        endwhile
        get #path, byt
        return
    else
        dat(i,2) = ys
        dat(i,5) = 0
        tempy = ys - 1
        while tempy <> ys do
            get #path, dummy2
            get #path, tempx
            get #path, tempy
        endwhile
        dat(i,3) = ( xs - tempx ) / 2
        dat(i,1) = xs - dat(i,3)
        tempy = ys - 1
        while tempy <> ys do
            get #path, dummy2
            get #path, tempx
            get #path, tempy
        endwhile
        get #path, byt
        return
    endif
end
```

（ＦＭ－11用変更点）

```
PROCEDURE draw_connect
(*    connect routine    *)
base 0
param num: integer
param dat(300,8): integer
dim max, xs, ys, xe, ye, co, fu: integer
dim ch: string[1]

    max = 299
    xs = 320
    ys = 100
    co = 7
    fu = 0
    run gcursor( xs, ys, xs, ys )
    loop
        run pset( xs, ys, 7, 4 )
        run gcursor( xs, ys, xe, ye )
        run loc023
        run input_color( co )
        run input_function( fu )
        run pset( xs, ys, 7, 4 )
        run line( xs, ys, xe, ye, fu, co )
        dat( num, 0 ) = 1
        dat( num, 1 ) = xs
        dat( num, 2 ) = ys
        dat( num, 3 ) = xe
        dat( num, 4 ) = ye
        dat( num, 5 ) = fu
        dat( num, 6 ) = co
        dat( num, 7 ) = 0
        num = num + 1
        if num > max then
            end
        endif
        xs = xe
        ys = ye
        run loc023
        print "cr)end else)continue";
        get #0, ch
        if ch = chr$(13) then
            end
        endif
    endloop
end



PROCEDURE draw_circle
(*    circle routine    *)
base 0
param num: integer
param dat(300.8): integer
dim max, xs, ys, xe, ye, r, co, fu, fill: integer
dim ch: string[1]

    max = 299
    xs = 320
    ys = 100
    co = 7
    fu = 0
    fill = 0
    loop
        run gcursor( xs, ys, xs, ys )
        run pset( xs. vs, 7, 4 )
        run gcursor( xs, ys, xe, ye )
        x = xs - xe
        y = fix( ( ys - ye ) / .4495 + .5 )
        r = fix( sqrt( x*x + y*y ) + .5 )
        run loc023
        run input_color( co )
        run input_function( fu )
        run input_fill( fill )
        run pset( xs, ys, 7, 4 )
        run circle( xs, ys, r, co, .4495, .0, 1., fill, fu )
        dat( num, 0 ) = 2
        dat( num, 1 ) = xs
        dat( num, 2 ) = ys
        dat( num, 3 ) = r
        dat( num, 4 ) = co
        dat( num, 5 ) = fill
        dat( num, 6 ) = fu
        num = num + 1
        if num > max then
            return
        endif
        run loc023
        print "cr)end else)continue";
        get #0, ch
        if ch = chr$(13) then
            end
        endif
    endloop
end


PROCEDURE draw_pset
(*    pset, preset routine    *)
base 0
param num: integer
param dat(300,8): integer
dim max, xs, ys, co, fu: integer
dim ch: string[1]

    max = 299
    xs = 320
    ys = 100
    co = 7
    fu = 0
    loop
        run gcursor( xs, ys, xs, ys )
        run input_function( fu )
        if fu <> 1 then
            run loc023
            run input_color( co )
        endif
        run pset( xs, ys, co, fu )
        dat( num, 0 ) = 4
        dat( num, 1 ) = xs
        dat( num, 2 ) = ys
        dat( num, 3 ) = co
        dat( num, 4 ) = fu
        num = num + 1
        if num > max then
            end
        endif
        run loc023
        print "cr)end else)continue";
        get #0, ch
        if ch = chr$(13) then
            end
        endif
    endloop
end


PROCEDURE loc023
(*    locate 0, 23    *)

    run line( 0, 184, 639, 199, 0, 0, 2 )
    run locate( 0, 23 )
    print "                                   ";
    print "                                    ";
    run locate( 0, 23 )
end
```

## 3－3－5　cards

●グラフィック機能を使った神経衰弱ゲーム

Basic09のグラフィック機能は、決して豊富とは言えませんが、そのグラフィックを用いた簡単なゲームプログラムを作成しました。

ＳＹＭＢＯＬ、ＫＰＲＩＮＴを含めて、Basic09で使えるグラフィック機能は、ＣＯＮＮＥＣＴを除き、すべて使っています。

ゲームの内容

トランプの「神経衰弱」は2人以上でプレイしますが、このプログラムは、1人でプレイするものです。

全てのプロシジャを、Basic09でロードして（メモリは１０Ｋバイト必要)、メインプログラム "cards"をＲＵＮすれば、ゲームは始まります。

または、あらかじめＰＡＣＫしておいて、Shell コマンドラインから cards を起動させます。

最初にオープニング画面が描かれ、制限時間を聞いてくるので、分単位で入力します。(3分から5分くらいが適当でしょう)

制限時間が入力されると、カードを全て裏返して、ランダムに並べます。

カードの裏側には、ちゃんと模様がついています。

準備が終わると左下に、"Ｘ　Ｙ："と表示され、カードの選択が要求されて、ゲームが開始されます。選んだカードの位置に従って、横・縦の順に、空白またはピリオド（カンマは使えません）で区切って入力します。入力が正しければ、そのカードが表に返され、入力に誤りがあれば、再入力を要求します。

さらに、もう一枚のカードを選んで、同様に入力すると、そのカードも表になって、2枚のカードの数を比べて、同じなら赤枠でそれらのカードを囲み、異なれば、また裏にもどします。

この時点で、経過時間のチェックをして、次のカードの選択へ進みます。

画面右上に、残りの枚数、経過時間、制限時間が表示されていますので、時間内に何枚合わせられるか、挑戦してみて下さい。

## プログラムについて

### １）概要

このプログラムは、メインを含め、全体で１４のプロシジャによって構成されていて、その構造は図１に示すようになっています。また、グラフィック機能は表１のような内容で使用しています。

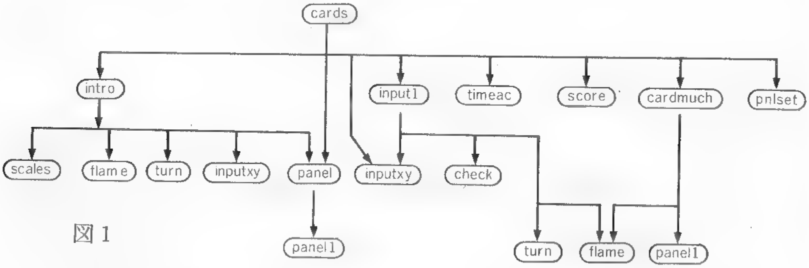


図１

カードは、その種類（マーク）と数に従って、３ケタの整数で表２のようにコード化して表わしています。そして、並べた場所の横位置、縦位置に従って、１３×４の配列に、そのコードが入れられます。もちろん、カードを並べるのには、ＲＮＤ関数を使っています。これらは、プロシジャ“pnlset”で行っているところです。

表１　グラフィック機能

| 機能名 | 用途 | プロシジャ |
|---|---|---|
| ＫＰＲＩＮＴ | タイトル表示 | intro |
| ＬＩＮＥ | カードの枠 | flame |
| ＰＡＩＮＴ | カードの地色 | panel1 |
| ＳＹＭＢＯＬ | カードの文字 | turn |
| ＰＳＥＴ | カードの模様 | panel1 |

カードを表に返すのは、プロシジャ“turn”で行っています。先ほどの３ケタのコードに従って、マークと数をＳＹＭＢＯＬ文で書いています。４種類のマークは、ＡＳＣＩＩコードが順につながっているので、ＣＨＲ＄を使って、簡単に書けますが、数の方は、“Ａ”、“Ｊ”、“Ｑ”、“Ｋ”を含んでいるので、文字定数を使って

います。（リスト参照）

カードの裏側を描いているのは、プロシジャ“panel1”で、PsetとＦＯＲ～ＮＥＸＴループを組み合わせて、単純な模様を作っています。この部分を工夫すると、色々な模様が作れるので、やってみるとおもしろいでしょう。

表２　カードの表わし方

| コード | マーク |
|---|---|
| １０１～１０３ | ♤ |
| ２０１～２１３ | ♡ |
| ３０１～３１３ | ◇ |
| ４０１～４１３ | ♧ |

↓　　↓
“A”　～　“K”

グラフィック画面は、６４０×２００と６４０×４００に対応できるように、座標に関する変数を多くのプロシジャ間でパラメータとして共有しています。６４０×４００の時は、メインプロシジャ“cards”の、y00＝１を、y00＝２と換えるだけで他に変更は不要です。

２）各プロシジャの機能

○プロシジャ　cards

このプロシジャは、メインプログラムであり、ここで重要なポイントはゲーム全体の流れの制御にあります。プログラムリストを見ると、約４分の１はＲＵＮ文になっていることからもわかるでしょう。

このゲームは、内容としては、プログラムの流れもあまり複雑ではありません。

表３　グラフィック座標関連変数

ｘ００：カード表示位置ｘ座標中心値
ｙ００：縦ドット数／２００
ｙ０　：カード表示上限位置
ｗｘ　：カードの横ドット数
ｗｙ　：カードの縦ドット数
ｌｘ　：カードの表示枚数（横）１～13
ｌｙ　：　　〃　　　（縦）２ or ４

大ざっぱに見ると、①初期設定、②カードの選択、表示、③数が一致したか判定、③残り枚数、時間の処理、④終了判定　といった流れになっていて、②～④でメインループが構成されています。

もう一つ重要なポイントに、グラフィック座標に関する変数の取り扱いがあります。先に述べたように、変数y00を２にするだけで６４０×４００にできるばかりではなく、グラフィック画面の上下、左右の平行移動、カードの大きさの変更、また、使用するカードの枚数の変更も同様に、このメインプロシジャの初期設定の部分の代入文を変えるだけで簡単にできるようになっています。表3を参照して下さい。

このように、メインプロシジャの変数で、全てのプロシジャを統一的に制御できるようにするため、各プロシジャのパラメータが多くなっているのです。

○プロシジャ　intro

オープニング画面の表示と、制限時間の入力を行うプロシジャです。オープニング画面がいらない、という方は、このプロシジャを省いても結構ですが、何らかの方法で、制限時間tlimitを設定しなければなりません。
また、実際に画面表示をしているのは、更に下位のプロシジャです。

○プロシジャ　panel

lx＊ly枚のカードを裏返しに並べて描くためのプロシジャで、次のPanel1を呼んでいます。

○プロシジャ　panel1

nx＊nyで示される位置にカードを裏返すプロシジャです。

paint文の後が、模様を描いているところです。Ｆ－ＢＡＳＩＣ等のグラフィックのＧＥＴ、ＰＵＴがないので、毎回、一枚ずつこうして描いています。

○プロシジャ　pnlset

lx＊ly枚のカードを並べる。すなわち配列変数に、先に述べた3ケタのコードを、ランダムに割り当てています。

○プロシジャ　cardmuch

2枚のカードを比べて、一致した場合と、しなかった場合の処理をしていま

す。

○プロシジャ　turn

px、py で示される位置のカードを表向きに描きます。

ＳＹＭＢＯＬ文を使っていますが、コードからのマーク、数字への変換方法、文字定数の使い方等、参考になると思います。

○プロシジャ　timeac

ＤＡＴＥ＄の右側、すなわち、文字型の時刻から実数で秒に変換します。

○プロシジャ　inputxy

カード選択の、ｘ、ｙを入力するために作ったプロシジャですが、整数型、文字型の一変数の入力にも使用できます。

なぜ単純にＩＮＰＵＴを使わないかというと、入力ミスがあった時、改行して再入力要求の表示がでてしまい、せっかくのグラフィック画面をだいなしにしてしまうからです。

このゲームのキーボード入力は全て、このプロシジャを用いています。

使い方は、第２パラメータを次のようにして呼びます。

| | | |
|---|---|---|
| 整数型２変数 | ０以上 | |
| 整数型１変数 | －１ | 第１パラメータに入力データ |
| 文字列１変数 | －２ | 第５パラメータに入力データ |

ただし、２変数の場合、デリミタは、スペースとピリオドのみ有効で、カンマは使用できません。

以上が主なプロシジャです。他のプロシジャは細かい機能なので説明は省略します。なお、各プロシジャの関係を図示しますのでご参考にして下さい。

```
PROCEDURE cards
dim x0, y0, lx, ly, mx, my, nx, ny, wx, wy, x00, y00: integer
dim pnl(13,4), dummy: integer
dim count, flag, pc, pnla, pnlb, time, tlimit: integer
dim start, now: real
dim cstart, cnow: string[8]
dim yn: string[79]

    run cls
    run color(7)
    tlimit = 0
    run score( 0, 0, 7 )
    flag = 0

    y00 = 2
    x00 = 300
    wx = 38
    wy = 26*y00
    lx = 13
    ly = 4
    x0 = x00 - wx*lx/2 + 1
    y0 = 50*y00

rem ---------- restart -----------
    repeat

        run color(7)
        count = 0
        time = 0
        run intro( x0, y0, wx, wy, lx, ly, y00, tlimit )
        exitif tlimit = 0 then
        endexit

        run panel( x0, y0, wx, wy, lx, ly, y00 )
        run pnlset( pnl, lx, ly )
        run score( lx*ly-count*2, time, 7 )
        run locate( 60, 3 )
        print using "s11,i7^", "time limit ", tlimit*60

rem ---------- start ----------
        cstart = right$( date$, 8 )
        run timeac( cstart, start )

        repeat

            run input1( x0, y0, wx, wy, lx, ly, mx, my, pnl, 22, y00 )
            pnla = pnl( mx, my )
            pnl( mx, my ) = 0

            run input1( x0, y0, wx, wy, lx, ly, nx, ny, pnl, 23, y00 )
            pnl( mx, my ) = pnla
            pnla = mod( pnla, 100 )
            pnlb = mod( pnl(nx,ny), 100 )
            cnow = right$( date$, 8 )
            run timeac( cnow, now )
            time = now - start
            run score( lx*ly-count*2, time, 7 )

            exitif time > 60*tlimit then
            endexit

            run cardmuch(x0,y0,lx,ly,mx,my,nx,ny,wx,wy,pnl,pnla,pnlb,count,y00)

            run score( lx*ly-count*2, time, 7 )
            run locate( 0, 1 )
            print "            "
            print "            "

        until count = lx*ly/2
rem -------- all done ---------

        run score( lx*ly-count*2, time, 2 )

        yn = "go on?"
        run inputxy( dummy, -2, 6, 21, yn )
    until yn = "n" or yn = "n"

rem ---------- ending ----------
    run color(7)

end
```

```
PROCEDURE intro
param x0, y0, wx, wy, lx, ly, y00, tlimit: integer
dim nx, ny, num: integer

    while tlimit = 0 do
        run kprint( x0+150, y0-40*y00, $3f40, 6 )
        run kprint( x0+200, y0-40*y00, $3750, 6 )
        run kprint( x0+250, y0-40*y00, $3f6a, 6 )
        run kprint( x0+300, y0-40*y00, $3c65, 6 )

        for ny = 1 to ly
            for nx = 1 to lx
                run flame( nx, ny, x0, y0, wx, wy, 7, y00 )
                num = nx+100*ny
                run turn( nx, ny, num, x0, y0, wx, wy, y00 )
            next nx
        next ny

        tlimit = 1
    endwhile
    run scales( x0, y0, wx, wy, lx, ly, y00 )

    run locate( 6, 21 )
    print "                            "+chr$(5)
    run inputxy( tlimit, -1, 6, 20, "time limit (min.)=" )

end



PROCEDURE panel
param x0, y0, wx, wy, lx, ly, y00: integer
dim x(2), y(2), nx, ny, xx: integer

    for nx = 1 to lx
        for ny = 1 to ly
            run panel1( x0, y0, wx, wy, nx, ny, y00 )
        next ny
    next nx
end



PROCEDURE pnlset
param pnl(13,4), lx, ly: integer
dim mx, my, nx, ny, card(13,4): integer

rem --- initialize card(13,4) ---

    for nx = 1 to 13
        for ny = 1 to 4
            card( nx, ny ) = 0
        next ny
    next nx

    for nx = 1 to lx
        for ny = 1 to ly
            repeat
                my = rnd(ly-1)+1
                mx = rnd(lx-1)+1
            until card(mx,my) = 0
            card(mx,my) = card(mx,my)+1
            pnl(nx,ny) = my*100+mx
        next ny
    next nx
end
```

```
PROCEDURE cardmuch
param x0,y0,lx,ly,mx,my,nx,ny,wx,wy,pnl(13,4),pnla,pnlb,count,y00:integer

     if pnla = pnlb then
          count = count + 1
          run flame( mx, my, x0, y0, wx, wy, 4, y00 )
          run flame( nx, ny, x0, y0, wx, wy, 4, y00 )
          pnl( mx, my ) = 0
          pnl( nx, ny ) = 0
     else
          for wait = 1 to 1000*y00
          next wait
          run flame( mx, my, x0, y0, wx, wy, 8, y00 )
          run flame( nx, ny, x0, y0, wx, wy, 8, y00 )
          run panel1( x0, y0, wx, wy, mx, my, y00 )
          run panel1( x0, y0, wx, wy, nx, ny, y00 )
     endif
end



PROCEDURE timeac
param ctime: string; time: real

     time=3600*val(left$(ctime,2))+60*val(mid$(ctime,4,2))+val(right$(ctime,2))
end



PROCEDURE input1
param x0, y0, wx, wy, lx, ly, mx, my, pnl(13,4), py, y00: integer
dim flag: integer
dim prompt: string[4]

     prompt = "x y:"
     my = 0
     repeat
          run inputxy( mx, my, 6, py, prompt )
          run check( mx, my, lx, ly, pnl, flag )
     until flag = 0
     run flame( mx, my, x0, y0, wx, wy, 8, y00 )
     run turn( mx, my, pnl(mx,my), x0, y0, wx, wy, y00 )
end



PROCEDURE turn
param px, py, num, x0, y0, wx, wy, y00: integer
dim x(2), y(2), c: integer
dim cnum, numlist: string

     numlist = "a 2 3 4 5 6 7 8 9 10j q k "

     x(1)  = x0+(px-1)*wx
     x(2)  = x0+px*wx-2
     y(1)  = y0+(py-1)*wy
     y(2)  = y0+py*wy-2*y00

     run paint( x(1)+wx/2, y(1)+wy/2, 7, 2 )
     c = num/100
     cnum = mid$( numlist, mod(num,100)*2-1, 2 )
     run symbol( x(1)+2, y(1)+2*y00, cnum, 1, 1, 0, 0 )
     run symbol( x(2)-2, y(2)-2*y00, cnum, 1, 1, 0, 2 )
     cnum = chr$(231+num/100)
     run symbol( x(1)+8, y(1)+10*y00, cnum, 2, 1, c )
end



PROCEDURE flame
param fx, fy, x0, y0, wx, wy, pc, y00: integer
dim x(2), y(2), func: integer
```

```
    func = 4
    if pc = 8 then
        pc = 7
    else
        func = 0
    endif

    x(1) = x0+(fx-1)*wx
    x(2) = x0+fx*wx-2
    y(1) = y0+(fy-1)*wy
    y(2) = y0+fy*wy-2*y00
    run line( x(1), y(1), x(2), y(2), func, pc, 1 )
end



PROCEDURE check
param x, y, lx, ly, pnl(13,4), flag: integer

    flag = 0

    if x < 1 or x > lx or y < 1 or y > ly then
        flag = 1
    else if pnl(x,y) = 0 then
        flag = 1
    endif ¥ endif
end



PROCEDURE panel1
param x0, y0, wx, wy, nx, ny, y00: integer
dim x(2), y(2), i, j, k, l, c1, c2: integer

    x(1) = x0+(nx-1)*wx
    x(2) = x0+nx*wx-2
    y(1) = y0+(ny-1)*wy
    y(2) = y0+ny*wy-2*y00
    run line( x(1), y(1), x(2), y(2), 0, 5, 1 )
    run paint( x(1)+1, y(1)+1, 1, 5 )

    c1 = 6
    c2 = 4
    l = (x(1)+x(2))/2
    for i = x(1)+4 to x(2)-4 step 4

        if i > l then
            c1 = 4
            c2 = 6
        endif

        k = abs(l-i)/2+4
        for j = y(1)+1 to y(1)+1+k*y00 step 2*y00
            run pset( i, j, c1, 0 )
        next j
        for j = y(2)-1-k*y00 to y(2)-1 step 2*y00
            run pset( i, j, c2, 2 )
        next j
    next i
end



PROCEDURE scales
param x0, y0, wx, wy, lx, ly, y00: integer
dim nx, ny: integer

    for nx = 1 to lx
        run symbol( x0+10+(nx-1)*wx, y0-wy+17*y00, str$(nx), 1, 1, 6 )
    next nx
    for ny = 1 to ly
        run symbol( x0+lx*wx+17, y0+8*y00+(ny-1)*wy, str$(ny), 1, 2, 6, 3 )
    next ny
```

```
end



PROCEDURE score
param rest, time, pc: integer

    run color( pc )
    run locate( 60, 1 )
    print using "s11,i7>", "rest cards ", rest
    run locate( 60, 2 )
    print using "s11,i7^", "time       ", time
end



PROCEDURE inputxy
param x, y, lx, ly: integer; prompt: string
dim lp, lk: integer
dim inkey, lkey: string

    on error goto 10
    lp = len(prompt)

10 rem -----------------------
    run locate( lx, ly )
    run color( 7 )
    print prompt;
    print chr$(5)+chr$(5)
    run locate( lx+lp, ly )
    run color( 7 )
    input "", inkey
    if y = -2 then
        prompt = inkey
        x = 0
    else
        x = val( inkey )
    endif
    if y > -1 then
        repeat
            lkey = left$( inkey, 1 )
            lk = len( inkey )
            inkey = right$( inkey, lk-1 )
        until lkey < "0"
        y = val( inkey )
    endif
end
```

# Shellの拡張

# 4—1　コマンドおよびフィルタとしてのShell

## 4—1—1　フィルタとしてのShell

Shell はＯＳ-９というシステムの中にあって特に重要なプログラムモジュールです。Shell を通すことによって初めてＯＳとのコミュニケーションが可能になるのです。しかしながら UNIX およびＯＳ-９の Shell は、ＣＰ／ＭにおけるＣＣＰのようにシステムに組み込まれたものではなく、単なるユーザプログラムにすぎません。

バークレー版のＵＮＩＸには C-Shell という、より機能拡張された Shell があります。ユーザは標準 Shell と C-Shell のどちらを使うのも自由です。

Shell はコマンドインタープリタであって、コマンドとして与えられた文字列を解析していき、必要ならば何等かのプログラムを起動するという働きしかしません。ただし現在のＯＳ-９の Shell はインタープリタとしての機能を大幅に省いてあるため、Shell プロシジャでコマンドを作るのが難しくなっています。

さしあたって現 Shell の中身を覗いてみることにしましょう。

Shell は基本的には、１行入力して解釈し処理するという流れを、入力にＥＯＦ(end of file) を検出するまで繰り返します。　つまり Shell はフィルタに他なりません。ただしＯＳ－９の多くのフィルタがそうであるように、コマンドラインにパラメータを置くことによってコマンドとしても機能します。

Shell 上で、

```
OS9: shell parameter
```

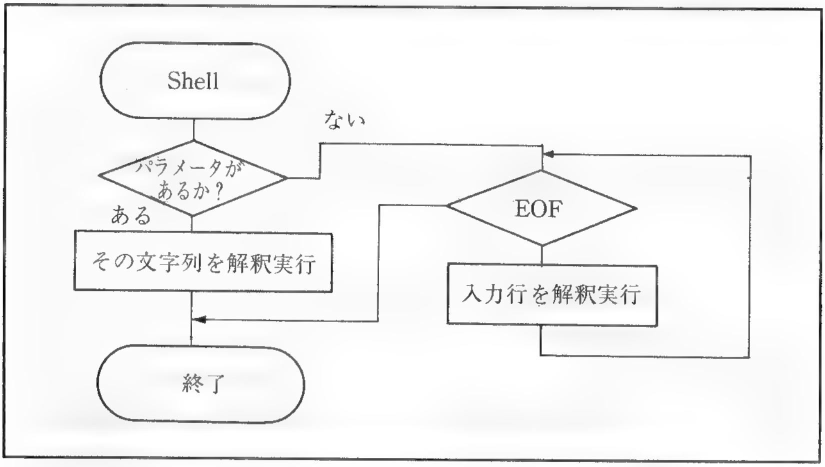
とすれば、“parameter”という文字列をパラメータとして(すなわちコマンドとして) 解釈実行します。この場合、“parameter”を処理した時点で終了してしまい、それ以上フィルタとして働くことはありません。

```
OS9 : shell
```

とパラメータを置かずに Shell を起動すると、すなわちフィルタとして機能しま

す。標準入力からＥＯＦまで読み込んでそれを処理していきます。

これは第１章から繰り返し述べてきたこととまったく同じです。



## 4－1－2　Shell オプション

Shell にもまたオプションがあります。オプションは他のパラメータとは区別され、Shell 自身の環境を変更するのに使われます。区別されるとは、

```
OS9 : shell p -x -t
```

のように、パラメータとして Shell オプションしか与えなかった場合、前節で述べたフィルタとして機能するということです。(p,-x,-t はすべて Shell オプションです。) したがってオプション以外のパラメータがあったかどうかを判定するためのフラグが１つ必要なのは言うまでもありません。

# 4－2　コマンドライン解釈の手順

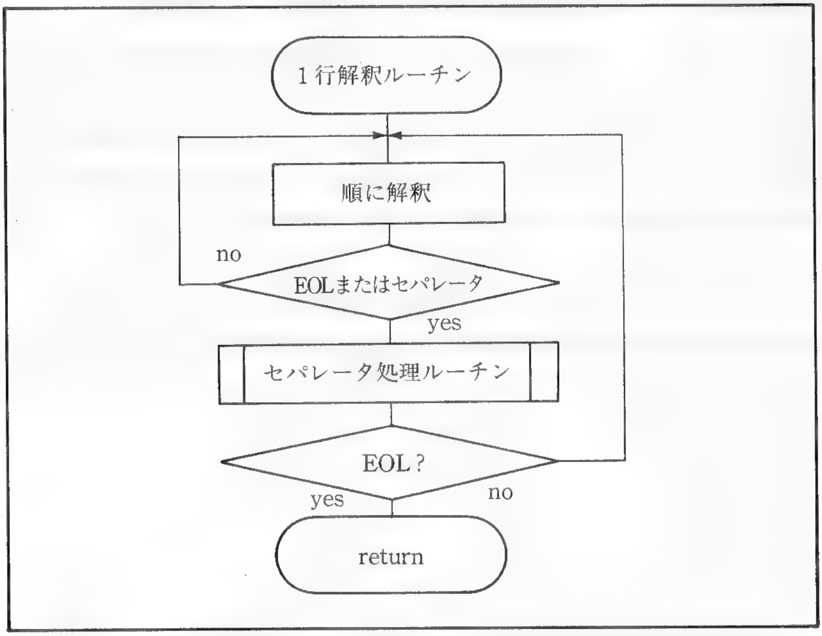
## 4－2－1　セパレータ

一般に１行のコマンドラインはセパレータで区切られた複数のコマンドから成ります。単一のコマンドを「単文」とするならこれは「重文」ということになります。例えば、

```
OS9: command1; command2 & command3
```

というコマンドラインが与えられた場合、

```
command1
command2
command3
```



という3つに分解されるわけです。この時セパレータは「単文」の間の関係を表します。関係というのは主に実行順序と標準入出力の2つの関係のことです。

一つ注意していただきたいのは、いかなる場合でもコマンドの解釈は左のものから順に行われるということです。それはセパレータの如何を問いません。

実際には、セパレータに行き当った時点で、そのセパレータの処理ルーチンに飛んでいます。

## 4—2—2　リダイレクト

コマンド起動時の指示の一つに標準入出力のリダイレクトがあります。

| | |
|---|---|
| ＜ | 標準入力 |
| ＞ | 標準出力 |
| ＞＞ | 標準エラー出力 |

コマンドを走査してゆく過程で上のような記号があった場合、リダイレクトしてそのコマンドを起動します。そのコマンドの標準入出力を切り替えるためにはShell自身の標準入出力を切り替え、その上でコマンドを起動するしかありません。またリダイレクトはそのコマンドだけに通用するもので、それ以後のコマンドにまで影響が出てはいけませんから、起動した直後に標準入出力をもとにもどしておかなければなりません。

リダイレクトすなわち標準入出力の切替えとは、ファイルをパス番号0ないし2に割り当てることです。しかし単にopenしただけではどんな番号が返されるか分かりません。強制的にパス番号0ないし2にするために、システムコールのF$Dupを使います。

標準出力をfileというファイルに割り当てる場合を考えましょう。これは次のような場合です。

```
command >file
```

まず、標準出別力のパス番号1を持ってF$Dupを発行します。すると何等かの

パス番号が返されます。それが3だったたとします。この3という数そのものにはたいした意味はなく、単なる退避用と考えて結構です。次にパス番号1をcloseします。

ファイルfileをcreate（またはopen）します。すると先程closeしたパス番号すなわち1のパスが割り当てられるのです。この理由は後で述べます。

以上で自分の標準出力はfileというファイルに割り当てられました。この時モジュールを起動すれば、標準入出力はそのままコピーされますからこれがすなわちリダイレクトに他なりません。

さてこのコマンドを起動し終った時点で、標準出力をもとにもどしておかないとそれ以降も（Shellが終了するまで）リダイレクトされたままになってしまいます。

そのために今と逆の手順を踏みます。

fileに切り替わっていたパス番号1をcloseします。次に先程Dupして得たところのパス番号3を持って再びF$Dupを発行します。するとたった今closeしたばかりのパス番号である1が返されます。以上で標準出力はもとにもどりました。

Dupのからくりを見てみましょう。

プロセスディスクリプタの中には１６バイトのパステーブルが用意されています。プロセスディスクリプタはプロセスの起動とともに生成され、終了とともに削除されます。ユーザがファイルをオープンした時に返されるパス番号とは、このパステーブルの中でのオフセットにすぎず、パステーブル中の各バイトが物理的なデバイスを表しているのです。

そして、ファイルをopenした時にはこのテーブルの中の小さい方から探してゆき、未使用の場所のパス番号を返すようになっているのです。F$Dupもまた同様に小さい方から探してゆきます。

最初、パステーブルが次のようになっていたとしましょう。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

ここで’＊’は何等かの物理的なパスを差しており、’0’は未使用であることを表します。0から2は標準入出力ですから通常はオープンした状態になっています。

この時パス1をF$Dupしますと、3が一番小さくて未使用ですから3が返されます。するとパス1と3は全く同じ物理デバイスを差すことになり、1を使っても3を使ってもまったく同じにそのファイルをアクセスできるでしょう。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ＊ | ＊ | ＊ | ＊ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

この状態でパス1をcloseします。すると1の場所に穴があきます。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ＊ | 0 | ＊ | ＊ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

ここでファイルfileをcreateします。するとまた小さいほうから探してゆき、パス1があいているのでパス1を返します。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ＊ | ＊ | ＊ | ＊ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

以上がリダイレクトの手順でした。次に元にもどす手順を見ます。

まずパス1で表されていたfileをcloseします。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ＊ | 0 | ＊ | ＊ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

これでパス1に穴があきます。最初にDupで得たパス3を持って再びF$Dupを発行しますと、小さい方から探していきますのでパス1が返されます。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ＊ | ＊ | ＊ | ＊ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

このパスは一番最初のパス1と全く同じ物理デバイスを表していることに注意して下さい。最後にパス3をcloseして完全に元に戻ります。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ＊ | ＊ | ＊ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 4—2—3 グルーピング

Shellにはセパレータで区切られた複数のコマンドをあたかも1つのコマンドであるかのように扱う、コマンドグルーピングという考え方があります。

グルーピングされたコマンドはさらに他のコマンドと組み合わせてグルーピングすることもできます。先の例で言うとこれは「複文」ということになりましょうか。例えば、

```
OS9 :(echo merge ; (wild ＊.c ! grep -v except.c ) ) ! join ! shell -p
```

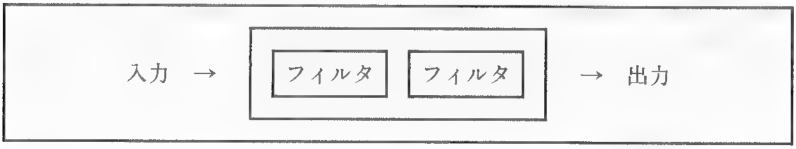
とすると、カレントディレクトリ中のCのソースファイルのうち、except.cを除いたすべてのファイルをマージします。

広い意味で、フィルタとは、データを入力して何等かの加工を施し、出力するものです。ただし、入力の無いものや出力の無いものもあります。

入力 → フィルタ → 出力

フィルタをセパレータで連結したものもまた、入力→加工→出力という形をとることから、それもまたフィルタに他ならないことが理解できるでしょう。この

場合のセパレータはパイプラインに限らず、シーケンシャルセパレータやコンカレントセパレータでも同じことです。



例えば、

```
OS9 : list file1 ; display c ; list file2
```

の場合、全体として見れば（入力はありませんが）ターミナルという一つのファイルへ何等かのデータを出力していることになります。

括弧でくくることによって、全体で１つのフィルタであることを表してみましょう。

```
OS9: ( list file1 ; display c ; list file2 ) >/p
```

先程は全体としてターミナルへ出力していたものが、このように括弧でくくることによって、全体としてプリンタに出力することになるのです。

では、この「全体として」とはどういう意味であるかと言いますと、括弧の中全体を一つのプロセスとして実行するということです。

結論から言うと、括弧の中身全体を一つのパラメータとして、Shellを起動してそれに渡します。その中がどうなっていようと、それは起動された側のShellの問題です。

Shellはコマンドを走査してゆく時に、コマンド名があるべき場所で'('に出会うと、そこをグルーピングの始まりと見なし、対応する')'が現れるまで左括弧と右括弧をカウントアップ、カウントダウンしてゆきます。そして対応する')'までの全体を一つの文字列として、Shellを起動してこれをパラメータとして与えます。

グルーピングに対するリダイレクトやデータエリアサイズの割当て、パイプラインなどはそのままShellに対して適用されます。

からくりはたったこれだけです。括弧がネスティングしている時は、その多重

度だけShellが多重に起動されるだけです。

新たにShellを起動するという点が重要である場合があります。内蔵コマンドを使用する場合がそうです。つまり内蔵コマンドはそのShellの中で有効なものですから、括弧の中に内蔵コマンドを置いた場合は、今のShellに結果が返って来ないのです。

```
OS9: (chd  /d0/usrdir  dir)
```

とした場合、chdは確かに実行されるのですが、それはグルーピングプロセスとして起動したShellの中で実行され、それの終結とともにchdした結果は消えてしまうのです。

逆に、この事実を利用することもできます。

```
OS9: dsave ! chd /d0/usr/userdir shell -p-t x
```

は、カレントディレクトリのファイルを/d0/usr/userdirへdsaveしますが、最後のchdの結果がコマンド実行終了後まで残ります。つまりこのコマンド終了時にはカレントディレクトリが、そのディレクトリに移動しています。これを避けたい場合、全体を括弧でくくればよいのです。

```
OS9: (dsave ! chd /d0/usr/userdir shell -p -t x)
```

こうすればコマンド終了後もカレントディレクトリは動いていません。

## 4—2—4　コマンドの形式

単一のコマンド（先の例での「単文」）を考えます。

一口にコマンドとかパラメータとか言っても、どのようにしてそれらをコマンドラインの中から見つけ出せばよいのでしょう。特にリダイレクトやデータエリアサイズの指定などがある場合、どこまでがパラメータであるかの判別は難しそうです。

まずセパレータというものは絶対的なもので、セパレータの位置は無条件でコマンドの区切りであることを認めることにしましょう。(これが実はパラメータと

してメタキャラクタを置けないという妨げになっているのですが。)

さてShellはコマンドラインの解釈にあたって、内蔵コマンドがあるかどうかを調べます。内蔵コマンドであればサブルーチンを呼び出して処理し、それを内蔵コマンドがなくなるまで続けます。この時点でその単コマンドが終了すれば次の単コマンドの処理に向かいます。内蔵コマンドの次にあるべきものは（セパレータでなければ）コマンド名です。コマンド名は一般にパスリストで与えられています。

コマンド名の次にはリダイレクトなどの指示（コマンド修飾子）であるかまたはパラメータです。まずコマンド修飾子であるかどうかを調べ、該当しなければパラメータです。

パラメータの範囲を調べるために、パラメータの先頭位置から予約キャラクタのサーチを始めます。つまり″！#&;<>″およびＥＯＬ（$0D）です。それが見つかったところ（－１）までがパラメータの範囲であったということになります。

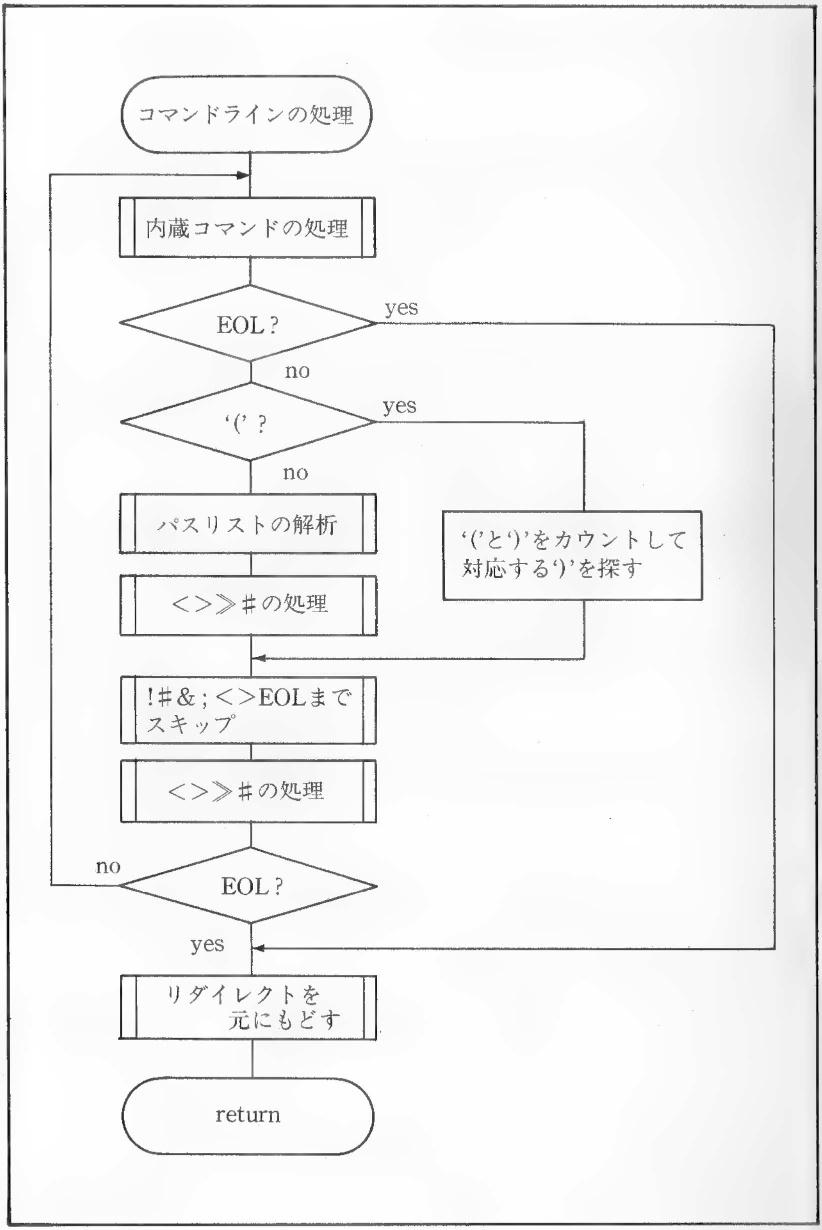
次に再びコマンド修飾子を調べます。

以上をまとめますと、

（内蔵コマンド）コマンド名　（修飾子）＜パラメータ＞　（修飾子）

′(′および′)′でくくってあるのは0個以上何個あってもよいことを表し′＜′および′＞′でくくってあるのは、あるか無いかのどちらかであることを表しているつもりです。

ひとつだけグルーピングという例外があります。コマンド名のあるべき位置に′(′があった時は無条件にグルーピングの始まりであると見なし、対応するべき′)′の位置まで、何が置いてあろうと関知しません。その中のコマンド修飾子等を処理するのは、新しく起動するShellであるはずなのです。


コマンドラインの処理
内蔵コマンドの処理
EOL？
yes
no
‘(’？
yes
no
パスリストの解析
‘(’と‘)’をカウントして対応する‘)’を探す
＜＞≫♯の処理
!♯&;＜＞EOLまでスキップ
＜＞≫♯の処理
no
EOL？
yes
リダイレクトを元にもどす
return

## 4－2－5　コマンドの取出し

モジュールを起動するために必要な情報は、

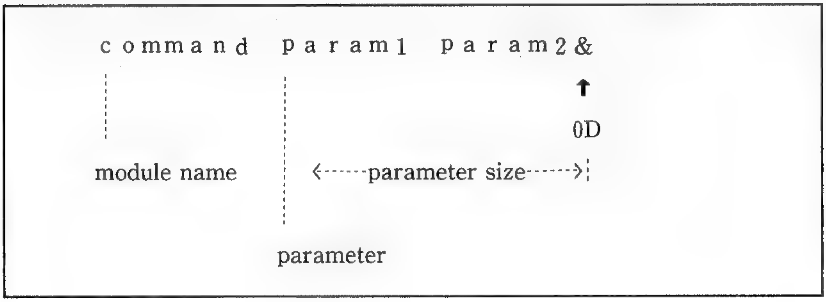
- モジュール名（またはパスリスト）
- パラメータ
- パラメータサイズ
- データエリアサイズ

の4つです。コマンドラインは全体で1つの文字列でしたから、コマンドラインの中からこれらの情報を取り出す作業が必要です。

なぜなら、パラメータはコマンドに渡された時には$0Dで終結した形になっている必要があります。一般にはモジュールをF$Forkで起動する際にはパラメータは必ずしも$0Dで終結していなくともよいのですが、Shellの場合に限っては必ず$0Dで終わるという約束があるのです。

もともとのコマンドラインには都合のよい位置に$0Dが置いてあるわけではないので、Shellがコマンドの解析時にしかるべき位置に$0Dを置きます。コマンドラインとは、1行読み込む場合は自分のバッファに読み込むわけですし、パラメータとして与えられた場合もやはり自分のものですから、そこにどんな加工を施しても構わないからです。

つまり一つのコマンドには次のように$0Dが埋め込まれるのです。（ただし内蔵コマンドの処理が終ってからです。）

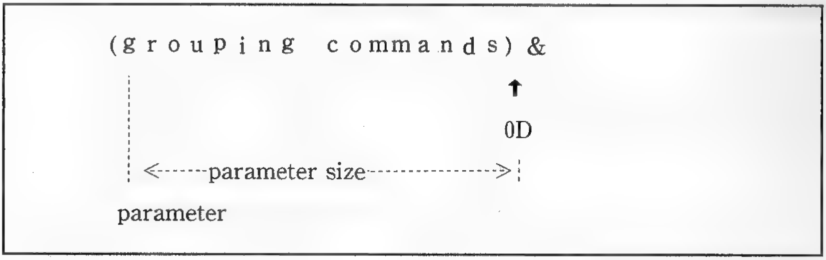


パラメータの位置は、commandをモジュール名（またはパスリスト）として解析（F$PrsNamを使う）して得られる、パスリストの次の位置から頭の空白をスキップした位置となります。

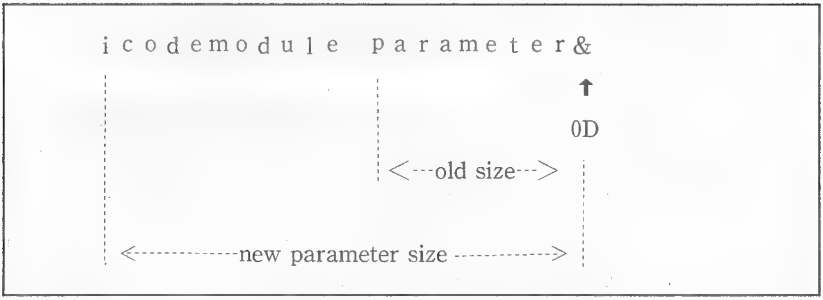
パラメータの終りは最初に識別子（;&! #＞＜）の現れた位置であると解釈します。上の例では'&'の位置がパラメータの終結であると見なされますが、その位置に$0Dを埋め込む前に'&'セパレータの処理がなされなければなりません。というより'&'セパレータを発見した時に$0Dを埋め込み、パラメータサイズを計算し、その後で'&'の処理ルーチンに飛ぶわけです。

グルーピングがあった場合は、上のcommandの位置に'('があるはずです。そして'（'の次の文字がパラメータの位置となり、パラメータの終りは対応する')'の位置になります。つまり')'の位置に$0Dが埋め込まれます。それより後ろはリダイレクトやセパレータなどの識別子でなければなりません。グルーピングの場合は起動すべきモジュールはShellですが、Shellという名前はモジュールヘッダにそのアドレスがありますので、それをそのまま使います。

なおグルーピングの場合は、括弧の中に何かがあるかは見ません。



これがもしBasic09のI―Codeモジュールのようにランタイムルーチンを必要とするものだと、モジュール名と思っていたものまでがパラメータとして扱わなければならない上、それは実際にLink（およびLoad）してみるまではわかりません。つまり後になってからモジュール名を変更することにもなります。モジュール名のアドレスと一緒にパラメータの位置とパラメータのサイズも変更しなければなりませんが、パラメータの位置はモジュール名だと思っていた位置をそのまま代入すればよく、パラメータサイズはパラメータの最後の位置からパラメータの位置（最初モジュール名だと思っていたもの）を引くことによって得ます。



例えばI—Codeモジュールの場合、パラメータは括弧でくくっておかなければならないことになっていますが、このようなことはランタイムルーチンの都合にすぎず、Shellはいっさい関知しません。

## 4－3　セパレータの処理

### 4－3－1　シーケンシャルセパレータ

記号';'はコマンドのシーケンシャルな実行を指定します。

コマンド走査の際に';'を見つけた時、およびセパレータ無しでＥＯＬになった時にシーケンシャルセパレータ処理ルーチンを呼び出します。

シーケンシャルとはそのコマンドの実行が終了するまで待つということです。つまり、コマンドのモジュールをF$Forkした直後にF$Waitするのです。

Forkしたモジュールが終了した時、リダイレクトした標準入出力を元にもどします。

### 4－3－2　コンカレントセパレータ

記号'&'はコマンドのコンカレントな実行を指定します。

コンカレントとは、コマンドをF$Forkしてすぐに次のコマンドの解析を始めるということです。

正常にForkできた場合はそのプロセスＩＤを“&００５”のように表示し、そしてリダイレクトした標準入出力をもとにもどして次の処理に移ります。

### 4－3－3　パイプラインセパレータ

記号'!'はパイプラインを指定します。

パイプラインは、'!'の左側のプロセスの標準出力と右側のプロセスの標準入力を接続します。

この２つのプロセスはコンカレントに実行されますが、”/pipe”というファイルを通して同期がとられます。

pipeファイルの実体はメモリ上の小さな（２５６バイト）バッファのことにすぎず、PipeManというファイルマネージャがこのバッファの管理をしています。

データを書き込んでいってこのバッファがいっぱいになった時、書き込んでい

たプロセスを（再びバッファが空くまで）眠らせてしまいます。データを読み出す方に対しては、もしデータが無かったらそのプロセスを（データが書き込まれるまで）眠らせてしまいます。もちろん1つのプロセスが読み出しと書き込みの両方をやったとしたら、ほどなく止ってしまうでしょう。つまりパイプは同時に走る2つのプロセスによってアクセスされなければならないのです。

パイプラインで重要な点は、各プロセスはそれぞれ標準入力、標準出力を使用するようにしなければならないということです。/pipeをオープンしただけではどんなパス番号が返されるか分かりませんから、リダイレクトの時と同様に、パス番号をすりかえる必要があります。それも、左側のプロセスは標準出力のすりかえを、右側のプロセスについては標準入力のすりかえを、と両方についてやらなければならないのです。つまりリダイレクトを2回繰り返します。

まずパス番号1を持ってF$Dupを発行します。すると例えば3という値が返されます。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

次に標準出力のパス番号であるパス1をF$Closeします。これでパステーブルの1の場所に穴があきます。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | 0 | * | * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

ここで"/pipe"をＵＰＤＡＴ.モードでF$Openします。パステーブルを小さい方から探していってパス番号1を返します。これでパス番号1でこのパイプをリード／ライトできます。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

これで左側の標準出力はパイプに接続されました。つまりその時点でのShell自身の標準出力はパイプになった訳ですから、この時プロセスを起動すればそれがそのまま引き継がれることになります。そこで左側のプロセスをF$Forkします。(Waitはしません。) 次は右側のプロセスです。

標準入力をすりかえるのですが、もし左側のプロセスがすでに標準入力をリダイレクトしていたとするとShell起動時のもともとの標準入力はすでにF$Dupされていて値が保存されているはずですから、再びF$Dupする必要はありません。そうでない場合だけF$Dupして標準入力を退避します。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

標準入力のパス番号0をF$Closeします。パステーブルの0の位置に穴があきます。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | * | * | * | * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

次にパス番号1をF$Dupします。パス1は先程のDupとOpenによりpipeを指しているはずです。当然パス0が返されますから、標準入力は同じパイプに接続されたことになります。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * | * | * | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

ここで、左側のプロセスの標準出力をリダイレクトしたものをもとにもどします。どうやってもとにもどすかは後で述べます。

この時点ではまだ右側のプロセスを起動しないことに注意して下さい。右側のプロセスを起動するのは、右側のプロセスの最後のセパレータ（またはＥＯＬ）に出会った時です。

さてリダイレクトしたパスを元にもどすには次のようにします。

あらかじめShellのグローバルな変数として3バイトを確保しておきます。それぞれが標準入力、標準出力、標準エラー出力に対応します。各バイトは0の時にはリダイレクトされていないことを表し、0でない時はF$Dupによって退避されたパス番号を表します。この値は今まで述べてきたリダイレクトの際に設定されます。例えば標準出力を元にもどす場合を考えましょう。

リダイレクトによってパス4に退避されているとしますと、3バイトのうち第1バイトには4という数値が入っています。

| 0 | 1 | 2 |
|---|---|---|
|  | 4 |  |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * |  | * |  |  |  |  |  |  |  |  |  |  |  |

パス1をF$Closeしますとパス1に穴があきます。

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | 0 | * |  | * |  |  |  |  |  |  |  |  |  |  |  |

退避していたパス4を持ってF$Dupするとパス1が返されます。そしてパス4をF$Closeすればパスはもとにもどります。もとにもどった印に3バイトのテーブルの1の場所をクリアします。

| 0 | 1 | 2 |
|---|---|---|
|  | 0 |  |

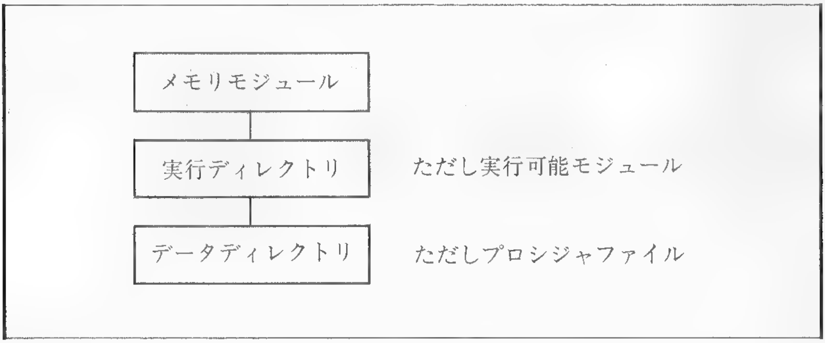
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| * | * | * |  | 0 |  |  |  |  |  |  |  |  |  |  |  |

# 4－4　コマンドの起動

## 4－4－1　モジュールのサーチの順序

Shellの働きはコマンドラインを解釈していって必要なモジュールを起動させることにあり、自分で何かをするということは（内蔵コマンドを除いて）ありません。

現Shellはモジュールを次の順序で探します。



完全なパスリストが与えられた場合は、実行可能モジュールであるかプロシジャファイルであるかを問わず、直接そのディレクトリを調べにいきます。

モジュールを起動するのはコマンドのパスリストを見た時でなく、セパレータを処理する時です。それまでは、パスリストの置いてある位置を覚えておきます。

モジュールのサーチは実際には次のように行われます。

まず、パスリストを持ってF$Linkしてみます。完全なパスリストだった場合はもちろん、モジュールディレクトリに存在しなければエラーになります。この場合はロードしなければなりません。

LinkできなかったFrom場合、アクセスモードをＥＸＥＣ.にしてF$Openしてみます。このようにすると実行ディレクトリの方でOpenされるのです。もちろん完全なパスリストだった場合はそのディレクトリでOpenされます。これでエラーが無ければヘッダの１３バイトを読み込み、F$Closeします。

実行ディレクトリでOpenできなかった場合データディレクトリでF$Openし

ます。この時はもはやモジュールとは認められず、プロシジャファイルであると解釈されます。したがってこのパスリストは起動すべきモジュール名ではなく、起動するべきモジュールはShellに他なりません。データディレクトリでOpenしたファイルはこのShellに対する標準入力として、前に述べたような手順でリダイレクトされます。

またこのShellにはパラメータとして”－P X ”を与えます。つまりプロシジャファイルの実行中にいちいちプロンプトを表示するなということ、および実行中にエラーが発生したらその時点で終了しろという指示です。

モジュールディレクトリにあった場合および実行ディレクトリにあった場合に話をもどします。ヘッダ中のタイプ／言語を見てPrgrm＋Objctだったならばそのままで実行できるということを意味しますが、例えばBasic09のI－Codeモジュールのように自分自身では実行できず、ランタイムルーチン（これはPrgrm＋Objctでなければなりません）を必要とするものがあります。

現在Shellがサポートしているランタイムルーチンは次のものです。

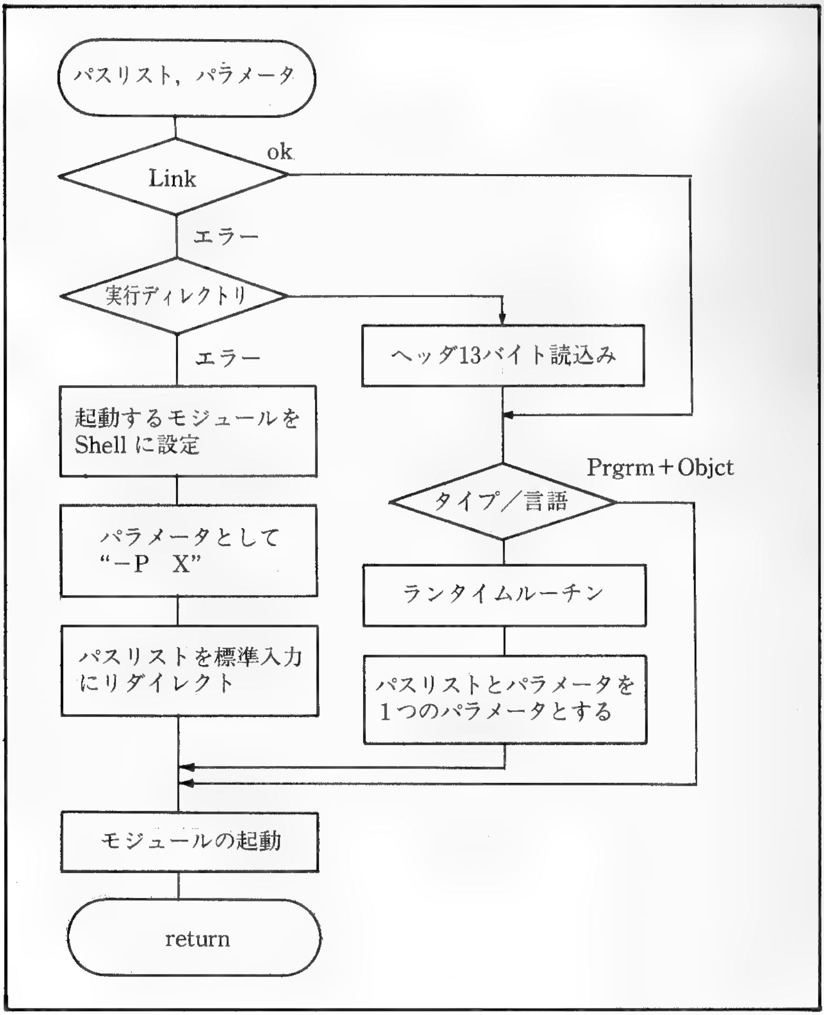
| タイプ／言語 | ランタイムルーチン |
|---|---|
| Prgrm＋PCode | PascalS（Pascal） |
| Sbrtn＋CblCode | RunC（Cobol） |
| Sbrtn＋ICode | RunB（Basic09） |

これらにタイプ／言語が一致すれば該当するランタイムルーチンの方をForkします。どれにも一致しなければ、実行できないというエラーになります。

この場合、パスリストそのものは実はランタイムルーチンに対するパラメータにすぎなかったということです。ではそもそもパラメータだと思っていたものは何だったかと言いますと、やはりパラメータだったのです。つまり、両方合わせたものがパラメータだったのです。

以上をまとめてみます。モジュール起動ルーチンは次の3つを得た後、実際にモジュールを起動します。

1．Fork するモジュール名
2．パラメータ
3．標準入力



モジュールディレクトリ上に存在した場合でも、存在を確かめただけですし、実行ディレクトリに発見された場合でもロードはしていなかったことを思い出して下さい。起動すべきモジュールが判明しましたのでひとまずF$Linkしてみます。エラーならばメモリには無いということなので改めてF$Loadします。これで初めてメモリ上に存在することが確定しましたのでF$Forkします。

## 4―4―2　内蔵コマンド

内蔵コマンドとShellオプションは結局のところ区別しても仕方ありませんので、一緒にして内蔵コマンドと呼ぶことにしましょう。

１つのコマンドについて、最初に内蔵コマンドがあるか無いかを調べ、処理ルーチンを呼び出します。これを内蔵コマンドがなくなるまで繰り返します。

なお内蔵コマンドは大文字と小文字の区別をしません。

内蔵コマンドを解説していきます。

| | |
|---|---|
| * | comment |

コマンドラインをＥＯＬまで読みとばし、何も実行しません。何もしませんがこれもりっぱなコマンドです。ライン上にセパレータがあろうと何があろうといっさいを無視します。ただし内蔵コマンドを置ける位置になければ意味をなさないことに注意して下さい。したがって内臓コマンド‘*’がある場合は、それより右側にはコマンドはありえないということになります。

| | |
|---|---|
| W | wait for child process to die |

システムコールF$Waitを発行して、子プロセスが終了するのを待ちます。子プロセスを一つも起動していなければエラーです。

また、正常に（いずれかの）子プロセスが終了したならば“－００５”のようにそのプロセス番号を表示します。

| | |
|---|---|
| CHD | change data directory |
| CHX | change execution directory |

CHD,CHXの次に置いたパスリストにI$ChgDirします。変更したものは、残りのコマンドのすべて（と言うよりそのShellが終了するまで）に有効です。

| | |
|---|---|
| P | prompt ON |
| −P | prompt OFF |

〝OS9:〟のプロンプトおよびEOF時の〝eof〟を出力するかどうかを指定します。これもShell終了まで有効です。

| | |
|---|---|
| T | echo back ON |
| −T | echo back OFF |

コマンドラインを（標準エラー出力に）エコーバックするかしないかを指定します。Shell終了時まで有効です。

| | |
|---|---|
| X | exit if error ON |
| −X | exit if error OFF |

コマンド解釈時または実行時にエラーがあった場合にShellをアボートするかしないかを指定します。これもShell終了時まで有効です。

| | |
|---|---|
| SETPR | set process priority |

プロセスのプライオリティを変更します。

SETPR　＜プロセス番号＞　＜プライオリティ＞

の書式で指定します。システムコール F$SPrior を発行するだけです。

## EX　exit and chain process

Shell 自身を終了させ、指定したモジュールを起動します。つまりシステムコール F$Chain を発行します。したがって指定できるのは実行可能なメモリモジュールだけです。

ＥＸを処理した時点で自分自身を解放してしまいますので、コマンドラインの残りは無視されます。

## KILL　kill process

指定したプロセスに対してF$Kill シグナルを送ります。存在しないプロセスをＫＩＬＬしようとするとエラーを返しますし、プロセス１(SysGo)に対してＫＩＬＬしようとすると”ＷＨＡＴ？”のメッセージを出します。

ただし、シグナルを送るだけでそれ以上の面倒は見てくれません。

# 4－5　プロトタイプShell

## 4－5－1　パターンマッチ

今まで述べてきたことをもとにして、Shellのプロトタイプを作ってみます。まずShellの使い勝手をよくするためにワイルドカードの機能を組み込むことにします。

Shellにおけるワイルドカードとは、パラメータとして置かれた文字列の中にワイルドカードキャラクタが置かれていた場合に、それにマッチするすべてのファイル名で置き替えることです。したがってコマンドラインそのものの中に$0Dを埋め込むという方法では実現できず、他にバッファを用意しておく必要があります。

したがってここでは、コマンドラインの解析の際には、ワイルドカードのある無しにかかわらずパラメータをそのバッファにコピーしていき、ワイルドカードだったら適当な文字列に置き替えたものをコピーするという風にします。

またワイルドカードを含む文字列はそれぞれ空白で区切られたパスリストであると考えなければならないため、現Shellがパラメータを全体で1つのものとして扱いその中身をいっさい見なかったのに対し、その中身を分断して見なければならず、ワイルドカードを含まない単語についても手を加えてしまうかもしれません。

しかし、ここで作るものはあくまでプロトタイプですから少しくらいのことには目をつむることにしましょう。

ワイルドカードに関連するコマンドを前の章までにいくつか挙げてきましたが1つのファイル名のパターンマッチを調べるルーチンを先に作っておきましょう。前章でBasic09で作成しておいたものをCに書き直します。

```
/*    pattern matching for wild card     */

#include <stdio.h>

#define YES 1
#define NO  0

int match( s, p )     /*  string, pattern  */
```

```
    char *s, *p;
{
    char *brakette();
    int  c;

    for ( ; c = *p++; s++ )
        if ( c == '*' ) {
            do {
                if ( match( s, p ) )
                    return( YES );
            } while ( *s++ );
            return( NO );
        } else if ( *s )
            switch ( c ) {
            case '[':
                if ( ( p = brakette( p, tolower( *s ) ) ) == NULL )
                    return( NO );
                break;
            case '^':
                if ( *s == *p++ )
                    return( NO );
                break;
            case '?':
                break;
            default:
                if ( tolower( c ) != tolower( *s ) )
                    return( NO );
                break;
            }
        else
            return( NO );
    return( *s == '¥0' );
}


char *brakette( p, sc )
    register char *p;
    int  sc;
{
    int  mat, found, c;

    mat = ( *p != '^' )? YES: ( p++, NO );
    for ( found = NO; *p && *p != ']'; p++ )
        if ( p[1] == '-' &&  p[2] && p[2] != ']' ) {
            if ( sc >= tolower( *p ) && sc <= tolower( p[2] ) )
                found = YES;
            p += 2;
        } else if ( sc == tolower( *p ) )
            found = YES;
    return( ( mat ^ found )? NO: (*p ? ++p: p ) );
}
```

## 4－5－2　wsh

match.c を使った Shell のプロトタイプを作ります。名前は仮に wsh といておきます。

ここではコマンドの解釈は自分ではやらないことにし、必要ならば Shell を呼ぶことにします。つまりコマンドラインをワイルドカードで展開したもので置き替え、そっくりそのまま Shell に渡すことにします。

つまりコマンドの起動とは Shell を起動し、置き替えたコマンドラインをそっくり Shell に渡すことです。

自分でコマンドラインの解析をやらないため、一部使えない内蔵コマンドが出てくる他、次のような不都合が生じます。

1. chd,chx が使えない。（意味を成しません。）
2. コマンド名の位置にワイルドカードを置いたものも展開してしまう。
3. その他本来正しくない使用法をしているものも展開してしまう。

本来リダイレクトにワイルドカードを用いてはならないのはもちろんですが、間違ってワイルドカードを置いてしまった場合でも wsh は間違いとは認識しません。したがってその場合に展開されるコマンドラインは意味の無いものとなり、実行結果は意図したものとは違うものになってしまうでしょう。その点には十分注意して下さい。

wsh は Shell の置き換えを意図したものではないのです。以上の点に触れるような使い方をする場合はもとの Shell にもどって実行するか、または wsh の中から Shell を起動してやるべきことを済ませ、再び wsh にもどる、といった使い方をして下さい。

```
/*    wild card shell    */

#include <stdio.h>

#define YES 1
#define NO  0
#define ERR (-1)

char parbuffer[BUFSIZ];
char inbuf[BUFSIZ];
char pattern[100];


main()
{
    FILE *fp, *fopen();
    int  sherr;

    setbuf(,stderr, 0 );
    fprintf( stderr, "wshell¥n" );

    for ( ;; ) {
        fprintf( stderr, "¥n" );
        for ( ;; ) {
            fprintf( stderr, "WSH:" );
            if ( gets( inbuf ) == NULL ) {
                fprintf( stderr, "eof¥n" );
                return;
            }
            if ( *inbuf == '¥0' )
                continue;
            if ( ( sherr = cmdline() ) != NO )
                prerr( 2, sherr );
            break;
        }
    }
}

int cmdline()
{
    char *p, *q, *getpattern();
    int  c;

    *parbuffer = '¥0';
    p = inbuf;
    while ( *p ) {
        p = getpattern( p, pattern );
        if ( *pattern == '¥0' )
            break;
        if ( iswild( pattern ) )
            filematch( parbuffer, pattern );
        else {
            strcat( parbuffer, pattern );
            strcat( parbuffer, " " );
        }
    }
    if ( *parbuffer ) {
        return( forkshell( parbuffer ) );
    } else
        return( NO );
}



int iswild( p )
    char *p;
{
    int  c;

    while ( c = *p++ )
        if ( c == '[' || c == '*' || c == '?' )
            return( YES );
    return( NO );
}
```

```
char *getpattern( p, q )
    char *p, *q;
{
    char *skipspace();
    register char *s;
    int  c;

    for ( p=skipspace(p), *(s=q)='¥0'; (c=*p) && c != ' '; p++, s++ )
        *s = *p;
    *s = '¥0';
    return( p );
}



char *skipspace( p )
    char *p;
{
    while ( *p++ == ' ' )
        ;
    return( --p );
}



int forkshell( p )
    char *p;
{
    int  len, cid;
    int  status = 0;

    p[len=strlen(p)] = '¥n';
    p[len+1] = '¥0';
    if ( ( cid = os9fork( "shell", len+1, p, 1, 1, 0 ) ) == ERR )
        return( errno );
    wait( &status );
    return( status );
}
```

```
/*    file name matching    */

#include <stdio.h>
#define DIRSIZ 32

filematch( buf, pat )
    char *buf, *pat;
{
    char *q, *rindex();

    if ( ( q = rindex( pat, '/' ) ) && q != pat ) {
        *q = '¥0';
        fmatch( buf, pat, q+1 );
        *g = '/';
    } else
        fmatch( buf, ".", (*pat=='/')? pat+1: pat );
}



fmatch( buf, plist, pat )
    char *buf, *plist, *pat;
{
    FILE *fp, *fopen();
    char *shapename();
    long fseek();
    char file[DIRSIZ];

    if ( (fp = fopen( plist, "d" ) ) == NULL ) {
        prerr( errno );
        return;
    }
    fseek(fp, (long) (0x40), 0);
    while ( fread( file, DIRSIZ, 1, fp ) ) {
        if ( *file == '¥0' )
            continue;
        shapename( file );
        if ( match( file, pat ) ) {
            strcat( buf, " " );
            strcat( buf, file );
        }
    }
}


char *shapename( p )
    char *p;
{
    int  i;

    for ( i = 0; i < DIRSIZ-3; i++ )
        if ( p[i] & 0x80 ) {
            p[i] &= 0x7f;
            p[++i] = '¥0';
            return;
        }
    p[DIRSIZ-3] = '¥0';
    return( p );
}
```

## 4－5－3　拡張Shell

ここまででShellの概略は摑めたかと思います。そこで次はワイルドカードだけでなく、その他の機能も拡張したShellを作ってみることにしましょう。

まず何を拡張するかです。

1．ワイルドカード
2．コマンドに特殊文字を渡せるようにする。
3．プロシジャファイルにパラメータを渡せるようにする
4．コマンドのサーチの方法に柔軟性を持たせる。

1）ワイルドカード、特殊キャラクタ

ワイルドカードの組み込み方の概略はすでに実現しました。しかしコマンドラインを解析していないので、コマンド名を置くべき位置にワイルドカードを指定したものまで展開してしまうのを避けることができませんでした。

またコマンドに予約キャラクタを直接渡す手段もありませんでした。ここではコマンドラインを単語に分解しながら解析していくという方法をとることにします。単語の中に予約キャラクタが含まれていてもよいことにします。

つまり、コマンドラインから１単語ずつ取り出してくる関数をもとにするのです。

2）プロシジャファイルにパラメータを渡す

Shellプロシジャにパラメータを渡すことを考えます。具体的な方法はひとまずおいて、その仕様をまず考えてみましょう。

パラメータを参照するのにもっとも簡単な方法として、パラメータを’＄１’などの変数（文字列変数）で表すことにします。第一パラメータは＄１で第２パラメータは＄２です。第０パラメータは常に自分の名前を表すことにします。また実際のパラメータの数を越える指定をした時はＮＵＬＬにします。

例えば、testという名前のプロシジャが次のようだったとします。

```
echo  $1  $2  $3
```

これを起動するには、

```
$  test  abc  def  ghi
```

のようにします。('$'はプロンプト)。すると次の出力が得られるでしょう。

```
abc  def  ghi
```

3）コマンドのサーチ順

プロシジャファイルは実行ディレクトリに置いても実行できないことは述べました。逆に実行モジュールをデータディレクトリに置いてもまた、完全なパスリストを与えない限りは実行できません。これはかなり不便なので次のように改造します。

モジュールのサーチの順序は変えない。つまり、

1．モジュールディレクトリ
2．実行ディレクトリ
3．データディレクトリ

ただし、2、3においては実行モジュールであるかプロシジャであるかを問いません。

例えば、window2などは実行ディレクトリに置いておけばよいのです。

一見簡単そうですが、実はかなりの変更が必要になります。

さて、このShellの名前はP-Shellとします。(コマンド名はpsh) なお、かなり多くのファイルに分割してコンパイルします。これはプログラムの機能を適当に分割するためであると同時に、機能拡張およびデバッグを容易にするためです。というよりは、機能を増やしていく毎に関数が増え、その度にファイルが増えていったというのが実情です。

1つのファイルのサイズが大きくなるとコンパイルタイムも大きくなってしまいます。いくつかのファイルに分けておいて、さしあたってデバッグをするファイル以外のものはリロケータブルオブジェクトに落としておけばリンクするファイルが増えるだけで、コンパイルするファイルのサイズは小さいものとなります。

そうした方がデバッグが楽です。

では、メインルーチンから、機能拡張を中心にして説明することにします。

```
/*    prototype shell     */

/*    usage: psh <internal> (parameter).

      refer to parameter as '$1'. ( '$0' always means "psh" )
      replace wild card with match file name.

      internal command:
          p               prompt ON.
          -p                     OFF.
          t               terminal echo back ON.
          -t                                 OFF.
          x               error exit ON.
          -x                         OFF.
          *               comment.
          chd <pathlist>  change data directory.
          chx <pathlist>  change execution directory.
          pw              print data sirectory.
          px              print execution directory.
          w               wait for child process to die.
          kill <process>  kill process.
          setpr <process> <priority>  set process priority.
          ex <pathlist>   fork and exit. ( difficult to Shell )
*/

#include <stdio.h>

#define YES 1
#define NO  0
#define ERR (-1)
#define EQUAL 0

#define SIN  0
#define SOUT 1
#define SERR 2

char parameter[BUFSIZ*2];     /* module parameter */
char inbuf[BUFSIZ];           /* input buffer     */
char pattern[BUFSIZ];
char wpos[256];               /* data directory   */
char xpos[100]  ;             /* execution directory */
unsigned datsize = 0;         /* data area size to fork */
int  prmpt = YES;             /* prompt flag      */
int  termback = NO;           /* echo back flag   */
int  exiterr = NO;            /* error exit flag  */
int  sigcode = 0;             /* signal code      */
int  stdtbl[3] = { 0, 0, 0 }; /* redirect table   */
char *pnt;                    /* pointer to command line */
int  npar = 0;                /* number of arguments */
char **par;                   /* pointer to array of arguments */


intrupt( sig )
    int  sig;
{
    sigcode = sig;
}


main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();
    int  intrupt();
    char *index();
    int  count, sherr;
    int  i;
    setbuf( stderr, NULL );
    intercept( intrupt );

    if ( sherr = option( argc, argv ) )
        exit( sherr );
    --par;
    par[0] = argv[0];

    pwd( wpos, NO );
    pwd( xpos, YES );

    if ( prmpt )
        fprintf( stderr, "pshell¥n" );
    for ( ;; ) {
        if ( prmpt )
            fprintf( stderr, "¥n" );
        for ( ;; ) {
            if ( prmpt )
                fprintf( stderr, "$ " );
            if ( (count=readln( SIN,inbuf,sizeof(inbuf))) == ERR ) {
                if ( errno && exiterr )
                    exit( errno );
                else if ( errno ) {
                    prerr( SERR, errno );
                    continue;
                }
            } else if ( count == 0 ) {      /* EOF */
                if ( prmpt ) {
                    fprintf( stderr, "eof¥n" );
                    return;
                } else
                    return;
            }
            inbuf[count-1] = '¥0';
            if ( termback ) {
                fputs( inbuf, stderr );
                putc( '¥n', stderr );
            }
            if ( *inbuf == '¥0' )
                continue;
            if ( ( sherr = cmdline() ) != NO ) {
                prerr( SERR, sherr );
                if ( exiterr )
                    return;
            }
            break;
        }
    }
}



int option( argc, argv )
    int  argc;
    char **argv;
{
    int  cmderr;

    while ( --argc ) {
        pnt = *++argv;
        if ( ( cmderr = intcomm() ) == YES )
            continue;
        else if ( cmderr )
            return( cmderr );
        else
            break;
    }
    npar = argc;
    par = argv;
    return( NO );
}
```

概略は前節の wsh と変わりません。しかしオプションの処理というものが必要

になります。ここで言うオプションとは、特に'p'と't'と'x'のことを指しています。

この3つはそれぞれ'prmpt'、'termback'、'exiterr'というグローバルなフラグで表します。またそれぞれに初期値が与えられます。

つまり通常のコマンドとして起動する時には、プロンプトを表示し、入力行のエコーバックはせず、エラーが発生しても終了しないようになっています。これらは内蔵コマンドによって変更できます。

メインルーチンは2つのfor文で囲まれた形になっていますが、この中で1行ずつ入力し、解析していきます。これはオリジナルとほとんど同じです。

1つ決定的に違うのは、pshは1回限りのコマンドとしては使えないということです。つまりオリジナルのShellはオプションまたは内蔵コマンド以外のものをパラメータとして置けば、そのコマンドを実行するだけで終了してしまいますが、pshはパラメータとして何を置こうと1行ずつ入力していくモードとなります。

これはプロシジャにパラメータを渡すことができるようにするためです。プロシジャというのは実際に実行するのはpsh自身で、プロシジャそのものはそれに対する標準入力にすぎないのですが、そのプロシジャにパラメータを渡すということは、「パラメータをコマンドとして、それ1行だけ処理して終了」したのではまずいのです。

プロシジャの処理とは疑似的に次のように表すことができます。

```
psh   parameter   <   procedure_file
```

このparameterを処理して止めてしまったのではプロシジャにならないのです。

オリジナルのShellではプロシジャにパラメータを渡すことを考えていないので、parameterの部分にコマンドを置きません。コマンドを置かないのでこれはすなわちフィルタであるとShellは解釈し、標準入力から1行ずつ読み込んでいくわけです。

実はShellはプロシジャに対して"−P　X　"というパラメータを与えています。これによってプロシジャはプロンプトを出さず、エラーがあったらアボートするようになっているのです。これは重要なことです。

では次にプロシジャへのパラメータを解釈する方法です。これは単純にargcとargvを使うことにします。つまり第一パラメータはargv[1]、第二パラメータはargv[2]です。

ここでひとつ考えてみましょう。先程プロシジャには”−P　X　”というパラメータを与えることが重要であると述べたばかりですが、それはどこに置けばよいのでしょう。

ユーザがパラメータとして置いたものと区別してやらなければならなくなります。結論として、区別をするのは不可能とまでは言わないまでも難しいので、pshは区別をしません。

つまりパラメータの最初にあるオプションまたは内蔵コマンドは、$1、$2などの範囲からはずします。プロシジャを起動する際にはユーザが与えたパラメータの前に”−p　x　”というものを接続し、それをパラメータとしてpshを起動するることにします。

例えば先のtestで、

```
$ test t chd /d0 abc def ghi
```

とすると、pshには次のパラメータが与えられ、

```
−p x t chd /d0 abc defs ghi
```

$1、$2、$3として、

```
abc def ghi
```

というものが得られます。

この解釈をするのが関数optionです。optionは1個の内蔵コマンドを解釈する関数intcommを、内蔵コマンドが見つからなくなるまで呼び出します。この時解析すべき文字列としてargvを順に与えていきますが、最終的には内蔵コマンドでないものを指すことになります。それを$1とすればよいわけです。

パラメータを指すポインタ配列はグローバルにしておく必要があるため、argcとargvのままでなく、nparおよびparにコピーしておきます。

問題は第0パラメータですが、本来プロシジャ名自身を表すべきなのですが、

プロシジャの中からプロシジャ名を知る方法が無いため、どうしてもプロシジャ名にしようとするなら起動する側でオプションとして与えてやらなければなりません。しかしここではpshの名前を指すようにしておきました。

さてメインルーチンとしてはもう一つインタラプトのことを避けるわけにはいきません。通常のコマンドは実行途中でもコントロールEなどでアボートしてしまいますが、無条件にアボートしては困りますので何等かの割込みに対するトラップを用意しておかなければならないのです。これはオプション’x’によって制御されなければなりません。

アセンブラレベルではシステムコールF$Icptによってこれを実現しますがCではこれに相当する関数interceptを用います。インタラプトが発生した時の処理をする関数がintrptです。mainではintrptをinterceptへの引数として用いますので、あらかじめ、

```
int   intrpt()；
```

と宣言しておかなければなりません。

intrptはインタラプトのコードをグローバルな変数sigcodeに代入するだけです。returnで値を返しても意味が無いことに注意して下さい。

sigcodeの値はしかるべき時にチェックします。

なお、入力にfgetsでなくreadlnを使用しているのは、fgetsが例えば1回コントロールEを入れると以後正常に動作してくれないという症状を示したための処置です。

関数pwdについては後で説明します。

```
/*    get one pattern from 'pnt'    */

char *getpattern( q )
    char *q;
{
    extern char *pnt;
    char *pat, *skipspace();
    int  c, quate;

    pat = q;
    if((c=*(pnt=skipspace(pnt)))==';'||c=='&'||c=='!'||c=='<'||c=='('||c==')') {
        ++pnt;
        *q++ = c;
        *q = '¥0';
        return( pat );
    } else if ( c == '>' ) {
        *q++ = c;
        if ( pnt[1] == '>' ) {
            *q++ = '>';
            ++pnt;
        }
        ++pnt;
        *q = '¥0';
        return( pat );
    } else if ( ( quate=c) == '¥"' || quate == '¥'' ) {
        *q++ = quate;
        while ( (c=*++pnt) && c != quate )
            *q++ = *pnt;
        if ( c )
            pnt++;
        *q = '¥0';
    } else {
        while ( (c=*pnt++) && c!=' ' && c!=';' && c!='&' && c!='|' && c!='<'
                && c!='>' && c!='(' && c!=')' && c!='#' && c!='¥"' && c!='¥'' )
            *q++ = c;
        --pnt;
        *q = '¥0';
    }
    return( pat );
}


char *skipspace( p )
    register char *p;
{
    while ( *p++ == ' ' )
        ;
    return( --p );
}
```

getpattern はコマンドラインから１単語分を取り出す関数です。

通常は空白を区切りとみなしますが、ダブルクォートまたはシングルクォートがある場合は、再びそのクォートが現われるかまたは行末に至るまでを取り出します。この時最初のクォートも含めたものを返しますが、これは、グルーピングで括弧の中にある単語を調べる際に、クォートを取り除いてしまうわけにはいかないからです。

```
/*    analize one command line    */

#include <stdio.h>

#define YES 1
#define NO  0
#define ERR (-1)
#define EQUAL 0

#define SIN  0
#define SOUT 1
#define SERR 2


int cmdline()
{
    extern char parameter[], inbuf[], pattern[];
    extern int  sigcode, datsize;
    extern char *pnt;
    }
    if ( n ) {                                  /* brakette incorrect */
        fprintf( stderr, "WHAT? illegal N.¥n" );
        stdback();
        return( NO );
    }
    strcat( strcpy( stdline, parameter ), "¥n" );
    *parameter = '¥0';
    for ( ;; ) {
        p = pnt;
        c = *getpattern( pattern );
        if ( c=='¥0'||c==';'||c=='&'||c=='!'||c=='#'||c=='<'||c=='>' ) {
            pnt = p;
            break;
        }
        if ( ( c = *pattern ) != '¥"' && c != '¥'' && iswild( pattern ) )
            filematch( parameter, pattern );
        else
            setpattern( pattern, parameter );
    }
    if ( cmderr = redirect() ) {
        close( pn );
        stdback();
        return( cmderr );
    }
    if ( redirpath( SIN, 3, "/pipe" ) ) {
        fprintf( stderr, "WHAT? can't open PIPE.¥n" );
        stdback();
        return( NO );
    }
    pn = dup( SIN );
    if ( write( pn, stdline, strlen( stdline ) ) == ERR )
        fprintf( stderr, "error #%03d.¥n", errno );
    close( pn );

    strcpy( modname, "psh" );
    strcpy( stdline, parameter );
    strcat( strcat( strcpy( parameter, "-p -t x " ), stdline ), "¥n" );
    if ( cmderr = separater( getpattern( pattern ), modname, parameter ) ) {
        close( pn );
        stdback();
        return( cmderr );
    }
    stdoneback( SIN );
    return( NO );
}


#include <ctype.h>

setpattern( p, q )
    char *p, *q;
{
    extern int  npar;
    extern char **par;
    int  n, c;

    if ( ( c = *p ) == '¥"' || c == '¥'' )
        ++p;
    if ( *q )
        strcat( q, " " );
    q += strlen( q );
    while ( c = *p++ )
        if ( c == '¥¥' && *p != '¥0' ) {
            *q++ = *p;
            p++;
        } else if ( c == '$' && isdigit( *p ) ) {
            if ( ( n = axtoi( &p ) ) <= npar ) {
                strcpy( q, par[n] );
                q += strlen( q );
            }
        } else
            *q++ = c;
    *q = '¥0';
}
```

```
int axtoi( ap )
    char **ap;
{
    char *p;
    int  n, c;

    p = *ap;
    for ( n = 0; isdigit( c = *p ); p++ )
        n = n*10 + c - '0';
    *ap = p;
    return( n );
}



int iswild( p )
    char *p;
{
    int  c;

    while ( c = *p++ )
        if ( c == '[' || c == '*' || c == '?' )
            return( YES );
    return( NO );
}
```

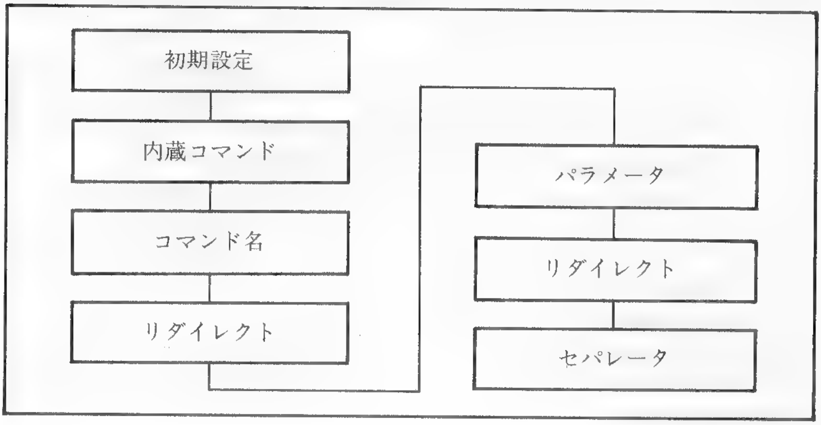
cmdline はコマンドラインの１行を解析する関数です。

１行の解析の手順はすでに述べました。ここでは for ループの中をその通りに解析しています。すなわち次頁の通りです。

１つのコマンドの解析を始めるにあたってまず、modname と parameter および datsize、sigcode をクリアします。

最初の３つはコマンドを起動する際に必要なもので、多くの関数から参照しますのでグローバルにしてあります。

sigcode は前に説明したインタラプトのコードが入るものです。



まず内蔵コマンドを処理します。関数 internal は pnt を見ていって内蔵コマンドがなくなるまで intcomm を呼び出して処理していきます。もし内蔵コマンドしかなかったのであればここで帰ります。

その次の単語はコマンド名でなければなりませんので、ただちに getpattern で modname にコマンド名を取り出します。もしこれが'（'であれば、それはグルーピングの始まりとみなします。

グルーピングでなかった場合、まずリダイレクトがあるかどうかを見ます。そして、リダイレクトかセパレータまたは行末が現われるまでがパラメータです。ここでは2種類の置き換えをしなければなりません。ワイルドカードとプロシジャパラメータです。どちらでもなければ取り出した単語を一つずつ parameter に接続していけばよいわけです。ただし間に空白を1つ詰めます。

まず単語の中にワイルドカード文字が含まれているかどうかを調べ、もしあれば filematch を呼び、なければ setpattern を呼びます。

filematch は与えられた文字列をワイルドカードで展開し、parameter に接続する関数であり、setpattern はプロシジャパラメータを含む文字列を展開してやはり parameter に接続する関数です。

パラメータの処理が終ったらリダイレクトの処理をするために関数 redirect を呼びます。redirect はリダイレクトの処理と'#'によるデータエリアサイズの指定を処理します。

最後にセパレータの処理関数である separater を呼びます。コマンドの起動はこの separater で行われます。

setpattern は文字列を1文字ずつ見ていって、もし'$'があればプロシジャパラメータと解釈して'$'の次の10進数を添字として par から文字列を取り出します。もしその範囲が npar より大きかった場合は無視します。

なお、setpattern でエスケープキャラクタ'¥'の解釈を行っています。

```
/*    separater    */

#include <stdio.h>

#define YES 1
#define NO  0
#define ERR (-1)

#define SIN  0
```

```
#define SOUT 1
#define SERR 2


int separater( sep, modname, parameter )
    char *sep, *modname, *parameter;
{
    if ( *sep == '¥0' || strcmp( sep, ";" ) == 0 ) {
        return( sequential( modname, parameter ) );
    } else if ( strcmp( sep, "&" ) == 0 ) {
        return( concurrent( modname, parameter ) );
    } else if ( strcmp( sep, "!" ) == 0 ) {
        return( pipeline( modname, parameter ) );
    } else
        return( ERR );



int sequential( modname, parameter )
    char *modname, *parameter;
{
    int  cid;
    int  seqerr = 0;

    if ( ( cid = forkcmd( modname, parameter ) ) == ERR )
        return( errno );
    stdback();
    seqerr = sigwait( cid );
    return( seqerr );
}



int concurrent( modname, parameter )
    char *modname, *parameter;
{
    int  cid;

    if ( ( cid = forkcmd( modname, parameter ) ) == ERR )
        return( errno );
    stdback();
    fprintf( stderr, "&%03d¥n", cid );
    return( NO );
}



int pipeline( modname, parameter )
    char *modname, *parameter;
{
    extern int stdtbl[];
    int  cid;

    if ( redirpath( SOUT, 3, "/pipe" ) ) {
        fprintf( stderr, "WHAT?¥n" );
        stdback();
        return( ERR );
    }
    if ( ( cid = forkcmd( modname, parameter ) ) == ERR )
       return( errno );
    if ( stdtbl[SIN] == 0 )
        stdtbl[SIN] = dup( SIN );     /* escape */
    close( SIN );
    dup( SOUT );
    stdoneback( SOUT );
    return( NO );
}



int sigwait( cid )      /* wait for cid die, if ( cid == 0 ) wait for any */
    int  cid;
{
    extern int sigcode;
    int  status = 0;
    int  xcid;
    for ( ;; ) {
        xcid = wait( &status );
        if ( !sigcode ) {
            if ( xcid == ERR || cid == xcid )
                return( status );
            else if ( cid == 0 || status ) {
                stdback();
                fprintf( stderr, "-%03d¥n", xcid );
                return( status );
            }
        } else if ( sigcode != 2 || cid == 0 )
            return( sigcode );
        else {
            kill( cid, sigcode );
            cid = 0;
        }
    }
}
```

実際にコマンドをForkするのはセパレータを解釈する時です。セパレータには3種類あって、それぞれに応じてsequential、concurrent、pipelineを呼び出します。コマンドのForkはさらに下請けの関数であるforkcmdにまかせます。sequentialとconcurrentでは、Forkした後でそのプロセスが終了するのを待つか待たないかの違いでしかありません。pipelineは標準入出力の切り換えを除けばconcurrentと似ています。

```
/*    fork command    */

#include <stdio.h>
#include <module.h>

#define YES 1
#define NO  0
#define ERR (-1)

#define SIN  0
#define SOUT 1
#define SERR 2


int forkcmd( modname, parameter )
    char *modname, *parameter;
{
    extern int  datsize;
    extern char wpos[], xpos[];
    char *modexec, *linkload();
    int  pn, cid;
    static char redfile[BUFSIZ];

    if ( *modname == '¥0' )
        return( ERR );
    strcat( parameter, "¥n" );

/* memory module */
    if ( ( modexec = modlink( modname, 0, 0 ) ) != ERR ) {
        if ( modexec[6] == 0x11 ) {
            cid = forkmodule( modname, parameter, datsize );
        } else if ( modexec[6] == 0x22 ) {
            strcpy( parameter, strcat( strcat( modname, " " ), parameter ) );
            cid = forkmodule( "runb", parameter, datsize );
        } else
            return( ERR );
        munlink( modexec );
        return( cid );
    }

/* execution or data directory */
    if ( ( pn = open( modname, 4 ) ) != ERR ) {
        if ( *modname != '/' )
            strcat( strcat( strcpy( redfile, xpos ), "/" ), modname );
    } else if ( ( pn = open( modname, 1 ) ) != ERR ) {
        if ( *modname != '/' )
            strcat( strcat( strcpy( redfile, wpos ), "/" ), modname );
    } else
        return( ERR );

    if ( ( modexec = linkload( pn, modname, parameter, redfile ) ) == ERR ) {
        close( pn );
        return( ERR );
    }
    close( pn );

     if ( *redfile == '¥0' ) {      /* module */
         cid = forkmodule( modname, parameter, datsize );
     } else {                             /* procedure */
         redirpath( SIN, 1, redfile );
         strcpy(parameter,strcat(strcpy(redfile,"-p -t x "),parameter));
         cid = forkmodule( modname, parameter, datsize );
         stdoneback( SIN );
     }
     munlink( modexec );
     return( cid );
 }



 char *linkload( pn, modname, parameter, redfile )
     int  pn;
     char *modname, *parameter, *redfile;
 {
     char *modexec;
     int  cid;
     struct {
         unsigned sync, size, name;
         char tyln, atrv, hp;
     } header;

     if ( read( pn, &header, sizeof(header) ) == ERR ) {
         close( pn );
         return( ERR );
     }
     if ( header.sync == 0x87cd && header.tyln == 0x11 ) {
         *redfile = '¥0';
         modexec = modload( modname, 1, 1 );
     } else if ( header.sync == 0x87cd && header.tyln == 0x22 ) {
         strcpy( parameter, strcat( strcat( modname, " " ), parameter ) );
         modexec = modlink( strcpy( modname, "runb" ), 0, 0 );
         *redfile = '¥0';
     } else if ( header.sync == 0x87cd )
         return( ERR );
     else {
         modexec = modlink( strcpy( modname, "wsh" ), 0, 0 );
     }
     return( modexec );
 }



 int forkmodule( modname, parameter, datsize )
     char *modname, *parameter;
     int  datsize;
 {
     int  cid;

     if ( (cid=os9fork(modname,strlen(parameter),parameter,1,1,datsize))==ERR )
         return( ERR );
     tsleep( 1 );
     return( cid );
 }
```

modname、parameter、datsizeをもとにして然るべきモジュールを起動させます。気をつけなければならないのは、modnameで与えられたコマンド名がそのまま起動すべきモジュール名とは限らないという点です。

まずmodnameをもとにモジュールディレクトリを探します。

さてここでまた難問があるのです。モジュールファイルであってもプロシジャファイルであってもまったく同じように扱いたいわけですが、モジュールファイルがデータディレクトリにある場合は、簡単には済まないのです。

つまりモジュールファイルは、モジュールディレクトリへロードしなければならないのに、システムコールF$Loadは実行ディレクトリに対してロードするようになっているのです。ただし完全なパスリストを与えた場合はこの限りではありませんが。

したがってLoadするためには完全なパスリストを作ってやらなければならないのです。これはpxdコマンドと同等の関数が必要だということです。

さて、Basic09のI-Codeのようにランタイムルーチンを必要とするモジュールだった場合には、modnameとparameterを接続したものを新たにパラメータparameterとしてランタイムルーチンの方をForkします。プロシジャファイルだった場合はmodnameのファイルを標準入力にリダイレクトしてpshをForkしますが、Parameterの前に”－p －t x ”を付けたものをパラメータとして渡します。このようにパラメータの前に勝手にオプションを付け加えてもよいのは、pshのメインルーチンをそのように設計したからです。

```
/*    get working directory    */

#include <stdio.h>

#define ERR (-1)

static char dir[32];
static int  exec;


char *pwd( p, ex )
    char *p;
    int  ex;
{
    FILE *fp, *fopen();
    char *getdev(), *pwdir(), *cdir();
    long parent, self;

    exec = ex;
    if ( ( fp = fopen( ".", cdir(exec) ) ) == NULL )
        exit( errno );
    fread( dir, sizeof(dir), 1, fp ); getinode( dir+29, &parent );
    fread( dir, sizeof(dir), 1, fp ); getinode( dir+29, &self );
    fclose( fp );
    change( "..", exec );
    *p = '¥0';
    pwdir( p, self );
    change( p, exec );
    return( p );
}


char *pwdir( p, myself )
    char *p;
    long myself;
{
    FILE *fp, *fopen();
    long parent, self, current;
    char *getdev(), shapename(), *cdir();

    if ( ( fp = fopen( ".", cdir( exec ) ) ) == NULL )
        exit( errno );
    fread( dir, sizeof(dir), 1, fp ); getinode( dir+29, &parent );
    fread( dir, sizeof(dir), 1, fp ); getinode( dir+29, &self );
    while ( fread( dir, sizeof(dir), 1, fp ) ) {
        if ( *dir == '¥0' )
            continue;
        getinode( dir+29, &current );
        if ( current == myself ) {
            shapename( dir );
            precat( p, dir );
            break;
        }
    }
    if ( parent == self )
        precat( p, getdev( fp, dir ) );
    else {
        change( "..", exec );
        p = pwdir( p, self );
    }
    fclose( fp );
    return( p );
}


precat( p, q )
    char *p, *q;
{
    static buf[BUFSIZ];

    if ( *p ) {
        strcat( strcat( strcpy( buf, "/" ), q ), p );
        strcpy( p, buf );
    } else
        strcat( strcpy( p, "/" ), q );
}


/*  getinode( p, own )   */
/*    char *p;           */
/*    long *own;         */
#asm
getinode:
 pshs x,y
 ldx 6,s
 ldy 8,s
 clr ,y
 lda ,x
 sta 1,y
 ldd 1,x
 std 2,y
 ldd #0
 puls x,y,pc
#endasm
```

```
#include <os9.h>

#define SS_DEVNM 14

char *getdev( fp, p )
    FILE *fp;
    char p[];
{
    struct registers reg;

    reg.rg_x = p;
    reg.rg_a = fileno( fp );
    reg.rg_b = SS_DEVNM;
    if ( _os9( I_GETSTT, &reg ) == ERR )
        exit( errno );
    shapename( p );
    return( p );
}


char *cdir( exec )
     int  exec;
{
    return( (exec)? "dx": "d" );
}



int change( p, exec )
     int  exec;
{
    return( (exec)? chxdir( p ): chdir( p ) );
}
```

カレントディレクトリの完全なパスリストを求めます。第2引数として0が置かれた場合はデータディレクトリを、それ以外の場合は実行ディレクトリを求めます。

完全なパスリストを求めるためには、カレントディレクトリから1レベルずつ上っていって、ディレクトリファイルを探していくしか方法がありません。

またルートディレクトリの名前を得るには直接ＯＳ－９のシステムコールである getstt を実行します。ファンクション番号１４（ＳＳ＿ＤＥＶNM）を与えると、パス番号で示されるディレクトリのデバイスディスクリプタ名を返してきます。

なおメインルーチンの最初にpwdを呼び出していたのは、コマンド起動の度にpwdを呼ぶのは面倒なので、常にカレントディレクトリ名を持っていたらどうかと考えたからなのですが、かえって煩しさが増したかもしれません。

```
/*    redirect    */

#include <stdio.h>
```

```
#define YES 1
#define NO  0
#define ERR (-1)
#define EQUAL 0

#define SIN  0
#define SOUT 1
#define SERR 2


int redirect()
{
    extern char *pnt, pattern[];
    char *p, *getpattern();
    int  rederr;

    for ( ;; ) {
        p = pnt;
        getpattern( pattern );
        if ( strcmp( pattern, "<" ) == EQUAL ) {
            if ( rederr = redirpath( SIN, 1, getpattern( pattern ) ) )
                return( rederr );
        } else if ( strcmp( pattern, ">" ) == EQUAL ) {
            if ( rederr = redirpath( SOUT, 2, getpattern( pattern ) ) )
                return( rederr );
        } else if ( strcmp( pattern, ">>" ) == EQUAL ) {
            if ( rederr = redirpath( SERR, 2, getpattern( pattern ) ) )
                return( rederr );
        } else if ( *pattern == '#' ) {
            if ( rederr = datchange( pattern+1 ) )
                return( rederr );
        } else {
            pnt = p;
            break;
        }
    }
    return( NO );
}



int redirpath( pn, mode, q )
    int  pn, mode;
    char *q;
{
    extern int stdtbl[];
    int  temp;

    if ( stdtbl[pn] ) {
        fprintf( stderr, "WHAT?¥n" );
        stdback();
        return( ERR );
    }
    if ( ( stdtbl[pn] = dup( pn ) ) == ERR ) {
        stdtbl[pn] = 0;
        stdback();
        return( errno );
    }
    close( pn );
    if ( pn == SIN && ( temp = open( q, mode ) ) != ERR )
        return( NO );
    else if ( ( pn == SOUT || pn == SERR )
               && ( (temp=creat(q,3))!=ERR || (temp=open(q,mode))!=ERR) )
        return( NO );
    else {
        temp = errno;
        stdback();
        return( temp );
    }
}



int stdback()
{
    int  pn;
```

```
        for ( pn = 0; pn <= 2; pn++ )
            stdoneback( pn );
        return( NO );
}


int stdoneback( pn )
        int  pn;
{
        extern int stdtbl[];

        if ( stdtbl[pn] ) {
            close( pn );
            dup( stdtbl[pn] );
            close( stdtbl[pn] );
            stdtbl[pn] = 0;
        }
        return( NO );
}


#include <ctype.h>

int datchange( p )
        char *p;
{
        extern int  datsize;
        int  c;

        if ( datsize || *p == '¥0' ) {
            fprintf( stderr, "WHAT?¥n" );
            return( ERR );
        }
        while ( c = *p++ )
            if ( isdigit( c ) )
                datsize = datsize*10 + c - '0';
            else if ( tolower( c ) == 'k' && *p == '¥0' )
                datsize <<= 2;
            else {
                fprintf( stderr, "WHAT?¥n" );
                return( ERR );
            }
        return( NO );
}
```

リダイレクトにはパス番号のすり換えが必要であることからfopenなどのファイルポインタによる入出力は行わずに、パス番号による入出力関数を使用しています。

またリダイレクトしたパス番号の退避用にstdtblという3要素からなる配列をグローバルに宣言してあります。getpatternによって得られた文字列が次のいずれかであるかによって引数をセットしてredirpathを呼びます。第一引数はリダイレクトすべきパス番号で、第二パラメータはリダイレクトするファイルをオープンする際のアクセスモード、第三パラメータはリダイレクトするファイルを表します。

全パスのリダイレクトをもとにもどすのは関数stdbackで、1つのパスについてもとにもどすのがstdonebackです。

リダイレクトの方法はすでに述べてありますのでここでは繰り返しません。

```
/*    internal commands    */

#include <stdio.h>

#define YES 1
#define NO  0
#define ERR (-1)
#define EQUAL 0


int internal()
{
    int  cmderr;

    while ( cmderr = intcomm() ) {
        if ( cmderr == YES )
            continue;
        else
            return( cmderr );
    }
    return( NO );
}



#include <signal.h>

int intcomm()
{
    extern char pattern[], wpos[], xpos[];
    extern char *pnt;
    extern int  stdtbl[];
    extern int  datsize, prmpt, termback, exiterr;
    char *p, *getpattern(), *lower();
    int  tid, pn, cmderr;

    p = pnt;
    lower( getpattern( pattern ) );
    if ( strcmp( pattern, "p" ) == EQUAL )
        prmpt = YES;
    else if ( strcmp( pattern, "-p" ) == EQUAL )
        prmpt = NO;
    else if ( strcmp( pattern, "t") == EQUAL )
        termback = YES;
    else if ( strcmp( pattern, "-t" ) == EQUAL )
        termback = NO;
    else if ( strcmp( pattern, "x" ) == EQUAL )
        exiterr = YES;
    else if ( strcmp( pattern, "-x" ) == EQUAL )
        exiterr = NO;
    else if ( strcmp( pattern, "pw" ) == EQUAL )
        fputs( wpos, stderr );
    else if ( strcmp( pattern, "px" ) == EQUAL )
        fputs( xpos, stderr );
    else if ( *pattern == '*' ) {
        while ( *pnt++ )
            ;

        --pnt;
    } else if ( strcmp( pattern, "w" ) == EQUAL ) {
        if ( cmderr = sigwait( 0 ) )
            return( cmderr );
    } else if ( strcmp( pattern, "chd" ) == EQUAL ) {
        if ( chdir( getpattern( pattern ) ) == ERR )
            return( errno );
        if ( *pattern == '/' )
            strcpy( wpos, pattern );
        else
            pwd( wpos, NO );
    } else if ( strcmp( pattern, "chx" ) == EQUAL ) {
        if ( chxdir( getpattern( pattern ) ) == ERR )
            return( errno );
        if ( *pattern == '/' )
            strcpy( xpos, pattern );
        else
            pwd( xpos, YES );
    } else if ( strcmp( pattern, "kill" ) == EQUAL ) {
        if ( ( tid = atoi( getpattern( pattern ) ) ) <= 1 ) {
            fprintf( stderr, "WHAT?¥n" );
            return( NO );
        } else if ( kill( tid, SIGKILL ) == ERR )
            return( errno );
    } else if ( strcmp( pattern, "setpr" ) == EQUAL ) {
        tid = atoi( getpattern( pattern ) );
        if ( setpr( tid, atoi( getpattern( pattern ) ) ) == ERR )
            return( errno );
    } else if ( strcmp( pattern, "ex" ) == EQUAL ) {
        if ( *getpattern( pattern) == '¥0' || cmderr == redirect() )
            return( cmderr );
        if ( modlink( p, 1, 1 ) == ERR )
            return( errno );
        for ( pn = 0; pn <= 2; pn++ )
            if ( stdtbl[pn] ) {
                close( stdtbl[pn] );
                stdtbl[pn] = 0;
            }
        strcat( pnt=skipspace(pnt), "¥n" );
        chain( p, strlen(pnt), pnt, 1, 1, datsize );
        exit( errno );
    } else {
        pnt = p;
        return( NO );     /*  not an internal command  */
    }
    return( YES );
}


char *lower( p )
    char *p;
{
    register char *q;
    int  c;

    for ( q = p; c = *q; q++ )
        *q = tolower( c );
    return( p );
}
```

１つの内蔵コマンドを処理するのがintcommで、内蔵コマンドがあるだけ処理するというのがinternalです。intcommはやや大きい関数になってしまいましたが、中身は単なるif－elseですから理解の妨げにはならないでしょう。

オリジナルのShellとほとんど同じですが、ＣＨＤのあたりに手を加えてあります。前述のように、カレントディレクトリをいつも覚えていようとしているためで、どこかに移る度にpwdを呼び出します。ただし、完全なパスリストでＣＨＤ、ＣＨＸを実行する場合にはそのパスリストをそのままコピーしてしまえばよいのです。

なお、カレントディレクトリを覚えているので、内蔵コマンドとしてpwおよびpxというものを追加してあります。働きはpwd、pxdコマンドと一緒ですが標準エラーに書き出しますので、他のコマンドの入力として使うことはできません。

# 付　録

## mcat

mcat はモジュールディレクトリに対しての cat です。出力をリダイレクトしないとファイルになりませんので注意して下さい。

```
/*    mcat  --  save memory module(s)     */

/*    usage: mcat {module}.    */


#include <stdio.h>
#include <module.h>

#define ERR (-1)
#define ANY  0
#define SOUT 1


main( argc, argv )
    int  argc;
    char **argv;
{
    char *p;
    mod_exec *header;

    while ( --argc ) {
        if ( ( header = modlink( p=*++argv, ANY, ANY ) ) == ERR ) {
            fprintf( stderr, "can't link ¥"%-s¥".¥n", p );
            return;
        }
        if ( write( SOUT, header, header->m_size ) == ERR )
            exit( errno );
    }
}
```

## chmod

attr コマンドはワイルドカード用に作られていないので、chmod を使って下さい。なおオプションはすべて‘－’で始めます。

```
/*    chmod  --  look or set file attribute     */

/*    usage: chmod <opt> {file} <opt>.

        opt: -x      : execution directory.
             -r  -nr : owner read.
             -w  -nw : owner write.
             -e  -ne : owner exec.
             -pr -npr: pablic read.
             -pw -npw: pablic write.
             -pe -npe: pablic exec.
             -s  -ns : sharable.
             -d  -nd : directory.
             -a      : suppress priting.    */
```

```
#include <stdio.h>
#include <modes.h>
#include <os9.h>

#define YES 1
#define NO  0
#define ERR (-1)
#define SUPERUSER 0
#define DTRBF 1

struct {
    char dtp;
    char dum1[18];
    char att;
    char fd[3];
    char dum2[9];
} pd;

struct {
    char perm;
    int  owner;
} fdes;

char root[32];
int  exec = NO;
int  suppress = NO;
int  cperm = 0;
int  nperm = 0;


main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fp, *fopen();
    char *p, *getroot();
    int  change, pn;
    union {
        long lpos;
        char cpos[4];
    } fdpos;

    if ( ( change = option( argc, argv ) ) == ERR ) {
        usage();
        return;
    }
    while ( --argc ) {
        if ( *(p=*++argv) == '-' )
            continue;
        if ( getoption( p ) == ERR )
            exit( errno );
        if ( pd.dtp != DTRBF )
            exit( 214 );
        if ( ( fp = fopen( getroot( root, p ), "r+" ) ) == NULL ) {
            fprintf( stderr, "can't open ¥"%-s¥".¥n", root );
            return;
        }
        fdpos.cpos[0] = pd.fd[0];
        fdpos.cpos[1] = pd.fd[1];
        fdpos.cpos[2] = pd.fd[2];
        fdpos.cpos[3] = 0;
        fseek( fp, fdpos.lpos, 0 );
        if ( fread( &fdes, sizeof(fdes), 1, fp ) == NULL )
            exit( errno );
        if ( (cperm&S_IFDIR) && !(fdes.perm&S_IFDIR) ) {
            fprintf( stderr, "can't change to directory.¥n" );
            fclose( fp );
            return;
        }
        if ( change ) {
            if ( (nperm&S_IFDIR) && (fdes.perm&S_IFDIR) && ownfile( p ) ) {
                fprintf( stderr, "the directory is not empty.¥n" );
                fclose( fp );
                return;
            }
            if ( changeperm( fp, fdpos.lpos ) == ERR ) {
                fclose( fp );
                return;
            }
        }
        fclose( fp );
        if ( !suppress )
            putmode( fdes.perm&0xff, p );
    }
}



int option( argc, argv )
    int  argc;
    char **argv;
{
    char *p;
    int  c, not, mode;
    int  change = NO;

    while ( --argc ) {
        if ( *(p=*++argv) != '-' )
            continue;
        if ( ( c = tolower( *++p ) ) == 'a' ) {
            suppress = YES;
            continue;
        } else if ( c == 'x' ) {
            exec = YES;
            continue;
        } else
            change = YES;
        if ( c == 'n' ) {
            not = YES;
            c = tolower( *++p );
        } else
            not = NO;
        switch ( c ) {
        case 'r': mode = S_IREAD;  break;
        case 'w': mode = S_IWRITE; break;
        case 'e': mode = S_IEXEC;  break;
        case 'p':
            switch ( c = tolower( *++p ) ) {
            case 'r': mode = S_IOREAD;  break;
            case 'w': mode = S_IOWRITE; break;
            case 'e': mode = S_IOEXEC;  break;
            default:  return( ERR );
            }
            break;
        case 's': mode = S_ISHARE; break;
        case 'd': mode = S_IFDIR;  break;
        default: return( ERR );
        }
        if ( *++p )
            return( ERR );
        else if ( not )
            nperm |= mode;
        else
            cperm |= mode;
    }
    return( change );
}



usage()
{
    fprintf( stderr, "usage: chmod <opt> {file} <opt>.¥n" );
    fprintf( stderr, "  opt: -x      : execution directory.¥n" );
    fprintf( stderr, "       -a      : suppress priting.¥n" );
    fprintf( stderr, "       -r  -nr : owner read.¥n" );
    fprintf( stderr, "       -w  -nw : owner write.¥n" );
    fprintf( stderr, "       -e  -ne : owner exec.¥n" );
    fprintf( stderr, "       -pr -npr: pablic read.¥n" );
    fprintf( stderr, "       -pw -npw: pablic write.¥n" );
    fprintf( stderr, "       -pe -npe: pablic exec.¥n" );
    fprintf( stderr, "       -s  -ns : sharable.¥n" );
    fprintf( stderr, "        -d  -nd : directory.¥n" );
}



char *getroot( root, p )
    char *root, *p;
{
    FILE *fp, *fopen();
    struct registers reg;
    char *q;
    int  c;

    q = root;
    if ( *p == '/' ) {
        while ( ( c = *++p ) && c != '/' )
            *q++ = c;
        *q++ = '@';
        *q = '¥0';
    } else if ( exec ) {
        if ( ( fp = fopen( ".", "dx" ) ) == NULL ) {
            fprintf( stderr, "can't open execution directory.¥n" );
            return( ERR );
        }
        reg.rg_x = root+1;
        reg.rg_a = fileno( fp );
        reg.rg_b = SS_DEVNM;
        if ( _os9( I_GETSTT, &reg ) == ERR ) {
            errno = reg.rg_b & 0xff;
            return( ERR );
        }
        fclose( fp );
        shapename( root+1 );
        *root = '/';
        strcat( root, "@" );
    } else
        strcpy( root, "@" );


    return( root );
}



shapename( p )
    char p[];
{
    int  i;

    for ( i = 0; i < 29; i++ )
        if ( p[i] & 0x80 ) {
            p[i] &= 0x7f;
            p[i+1] = '¥0';
            return;
        }
    p[29] = '¥0';
}



int getoption( p )
    char *p;
{
    int  pn, mode;

    mode = (exec)? S_IEXEC: 0;
    if ( (pn=open(p,mode)) == ERR && (pn=open(p,mode+S_IFDIR)) == ERR
                               && (pn=open(p,mode+S_IFDIR+S_IREAD)) == ERR )
        return( ERR );
    if ( getstat( SS_OPT, pn, &pd ) == ERR ) {
        close( pn );
        return( ERR );
    }
    close( pn );
    return( NO );
}
```

```
int ownfile( p )
    char *p;
{
    FILE *fp, *fopen();
    char file[32];

     if ( ( fp = fopen( p, "d" ) ) == NULL )
         exit( errno );
     fseek( fp, 64L, 0 );
     while ( fread( file, sizeof(file), 1, fp ) )
         if ( *file ) {
             fclose( fp );
             return( YES );
         }
     fclose( fp );
     return( NO );
}


int changeperm( fp, fdpos )
    FILE *fp;
    long fdpos;
{
    int  user;

    if ( ( user = getuid() ) != SUPERUSER && user != fdes.owner ) {
        fprintf( stderr, "you're not owner or superuser.¥n" );
        return( ERR );
    }
    fdes.perm = (fdes.perm|cperm)&‾nperm;
    fseek( fp, fdpos, 0 );
    if ( fwrite( &fdes, sizeof(fdes), 1, fp ) == NULL )
        return( ERR );
    return( NO );
}


putmode( perm, p )
    int  perm;
    char *p;
{
    char line[9];
    int  flag, i;

    strcpy( line, "dsewrewr");
    for ( flag = 1, i = 7; i >= 0; i--, flag <<= 1 )
        if ( !(perm&flag) )
            line[i] = '-';
    printf( "%s  %s¥n", line, p );
}
```

## ls

lsはdirと同じような機能を持ちます。デフォルトはディレクトリの詳細な情報を出さないようになっています。

```
OS9:ls
Motd
Errmsg
Dictionary
```

```
Tankanji
password
```

-e オプションを付けることにより詳細な情報を出力します。

この中で lnkという見慣れない欄があります。これはファイルのリンクカウントを表しています。通常のファイルはリンクカウントが1ですが、後で挙げる ln コマンドによって複数のリンクを持たせることが可能です。

```
OS9:ls -ef
   directory of . 17:11:57
 Owner  Last modified  attributes lnk sector bytecount name
------- -------------- ---------- --- ------ --------- ----------
  0.0   84/02/18 16:23  ----r-wr    1    371        10f Motd
  0.0   84/02/18 16:23  ----r--r    1    374        c7d Errmsg
  0.0   84/02/18 16:23  ----r--r    1    382         11 Dictionary
  0.0   84/07/24 02:57  ---wr-wr    1    384       6c86 Tankanji
  0.0   84/07/02 20:30  -------r    1    3f2         8b password
```

```
/*    ls --  list directory information    */

/*    usage: ls <opt> (file) <opt>.

        opt: -x     execution directory.
             -e     expand output.
             -f     format for header.    */

#include <stdio.h>
#include <time.h>


#define YES 1
#define NO  0
#define ERR (-1)


struct filedes {
    char att;
    char own[2];
    char date[5];
    char lnk;
    long siz;
    char cret[3];
};


int  exec = NO;
int  expand = NO;
int  format = NO;



main( argc, argv )
    int  argc;
    char **argv;
{
    char *p;
    int  exist;

    pflinit();
    if ( ( exist = option( argc, argv ) ) == ERR ) {
        usage();
        return;
    } else if ( exist )
        while ( --argc ) {
            if ( *(p=*++argv) == '-' )
                continue;
            if ( expand && format )
                dputheader( p );
            dls( p );
        }
    else {
        if ( expand && format )
            dputheader( "." );
        dls( "." );
    }
}



int option( argc, argv )
    int  argc;
    char **argv;
{
    char *p;
    int  exist = NO;
    int  c;

    while ( --argc ) {
        if ( *(p=*++argv) == '-' )
            while ( c = tolower( *++p ) )
                switch ( c ) {
                case 'x': exec = YES;   break;
                case 'e': expand = YES; break;
                case 'f': format = YES; break;
                default: return( ERR );
                }
        else
            exist = YES;
    }
    return( exist );
}



usage()
{
    fprintf( stderr, "usage: ls <-xef> {file|dir} <-xef>.¥n" );
}



dputheader( p )
    char *p;
{
    struct sgtbuf pack;

    getime( &pack );
    printf( "   directory of %-s %02d:%02d:%02d¥n",
                        p, pack.t_hour, pack.t_minute, pack.t_second);
    printf( " Owner  Last modified  attributes " );
    printf( "lnk sector bytecount name¥n" );
    printf( "------- -------------- ---------- " );
    printf( "--- ------ --------- ----------¥n" );
}



dls( p )
    char *p;
{
    static char root[32];
    static char dir[BUFSIZ];
    static char file[32];
    FILE *rfp, *fp, *fopen();
    long pos, getinode();

    dirfile( p, root, dir, file );
    if ( ( rfp = ( exec? fopen(root,"rx"): fopen(root,"r") ) ) == NULL ) {
        fprintf( stderr, "can't open ¥"%-s¥".¥n", root );
        return;
    }
    if ( fp = ( exec? fopen( p, "dx" ): fopen( p, "d" ) ) ) {
        lsdir( rfp, fp, p );
        fclose( fp );
    } else {
        if ( ( fp = ( exec? fopen(dir,"dx"): fopen(dir,"d") ) ) == NULL ) {
            fprintf( stderr, "can't open ¥"%-s¥".¥n", dir );
            return;
        }
        if ( ( pos = getinode( fp, file ) ) == ERR ) {
            fprintf( stderr, "can't open ¥"%-s¥".¥n", p );
            fclose( fp );
            return;
        }
        fclose( fp );
        putinform( file, pos, rfp );
    }
    fclose( rfp );
}



lsdir( rfp, fp, p )
    FILE *rfp, *fp;
    char *p;
{
    union {
        long lpos;
        char cpos[4];
    } pos;
    char *shapename();
    static char dir[32];

    fseek( fp, 64L, 0 );
    while ( fread( dir, sizeof(dir), 1, fp ) ) {
        if ( *dir == '¥0' )
            continue;
        pos.cpos[0] = dir[29];
        pos.cpos[1] = dir[30];
        pos.cpos[2] = dir[31];
        pos.cpos[3] = 0;
        shapename( dir );
        putinform( dir, pos.lpos, rfp );
    }
}



dirfile( p, root, dir, file )
    char *p, *root, *dir, *file;
{
    char *q, *rindex();

    *root = '¥0';
    if ( *p == '/' ) {
        if ( q = index( p+1, '/' ) ) {
            *q = '¥0';
            strcpy( root, p );
            *q = '/';
        } else
            strcpy( root, p );
    }
    strcat( root, "@" );
    if ( q = rindex( p, '/' ) ) {
        if ( p != q ) {
            *q = '¥0';
            strcpy( dir, p );
            strcpy( file, q+1 );
            *q = '/';
        } else {
            strcpy( dir, p );
            strcpy( file, "." );
        }
    } else {
        strcpy( dir, "." );
        strcpy( file, p );
    }
}


long getinode( fp, p )
    FILE *fp;
    char *p;
{
    static char dir[32];
    union {
        long lpos;
        char cpos[4];
    } pos;
    char *shapename();

    while ( fread( dir, sizeof(dir), 1, fp ) ) {
        if ( *dir == '¥0' )
            continue;
        pos.cpos[0] = dir[29];
        pos.cpos[1] = dir[30];
        pos.cpos[2] = dir[31];
        pos.cpos[3] = 0;
        if ( equal( shapename( dir ), p ) )
            return( pos.lpos );
    }
    return( ERR );
}


int equal( p, q )
    char *p, *q;
{
    int  c1, c2;

    for ( ; ( c1 = *p ) && ( c2 = *q ); p++, q++ )
        if ( tolower( c1 ) != tolower( c2 ) )
            return( NO );
    return( !*p && !*q );
}


char *shapename( p )
    char p[];
{
    int  i;

    for ( i = 0; i < 29; i++ )
        if ( p[i] & 0x80 ) {
            p[i] &= 0x7f;
            p[i+1] = '¥0';
            return( p );
        }
    p[29] = '¥0';
    return( p );
}


putinform( p, pos, rfp )
    char *p;
    long pos;
    FILE *rfp;
{
    static struct filedes fdes;
    static char line[9];
    int  i, mask;

    fseek( rfp, pos, 0 );
    if ( fread( &fdes, sizeof(fdes), 1, rfp ) == NULL ) {
        fprintf( stderr, "read error.¥n" );
        return;
    }
    if ( expand ) {
        strcpy( line, "dsewrewr" );
        for ( mask = 1, i = 7; i >= 0; i--, mask<<=1 )
            if ( !(fdes.att&mask) )
                line[i] = '-';
        printf( " %2d.%-2d  %02d/%02d/%02d %02d:%02d  %s   %2d %6lx  %8lx ",
                fdes.own[0]&0xff,  fdes.own[1]&0xff,
                fdes.date[0]&0xff, fdes.date[1]&0xff, fdes.date[2]&0xff,
                fdes.date[3]&0xff, fdes.date[4]&0xff, line,
                  fdes.lnk&0xff, (pos>>8)&0xffffff, fdes.siz );
    }
    printf( "%-s¥n", p );
}
```

## ln

リンクとは、１つのファイルの実体に対してのディレクトリエントリの数のことです。

```
OS9:ln tankanji temp

OS9:ls -ef
   directory of . 17:39:18
 Owner  Last modified  attributes lnk sector bytecount name
------- ------------- ---------- --- ------ --------- ----------
  0.0   84/02/18 16:23  ----r-wr   1    371       10f Motd
  0.0   84/02/18 16:23  ----r--r   1    374       c7d Errmsg
  0.0   84/02/18 16:23  ----r--r   1    382        11 Dictionary
  0.0   84/11/29 17:13  ---wr-wr   2    384      6c86 Tankanji
  0.0   84/07/02 20:30  -------r   1    3f2        8b password
  0.0   84/11/29 17:13  ---wr-wr   2    384      6c86 temp
```

**temp** というファイルに対して **temp2**という別の名前（ディレクトリエントリ）を付けてみます。

```
OS9:ln temp temp2

OS9:ls -e
  0.0   84/02/18 16:23  ----r-wr   1    371       10f Motd
  0.0   84/02/18 16:23  ----r--r   1    374       c7d Errmsg
  0.0   84/02/18 16:23  ----r--r   1    382        11 Dictionary
  0.0   84/11/29 17:13  ---wr-wr   3    384      6c86 Tankanji
  0.0   84/07/02 20:30  -------r   1    3f2        8b password
  0.0   84/11/29 17:13  ---wr-wr   3    384      6c86 temp
  0.0   84/11/29 17:13  ---wr-wr   3    384      6c86 temp2
```

リンクカウントが１増えて２になりました。

**temp** も**temp2** も同一のセクタ（ファイルディスクリプタ）を指しています。

**del** コマンドでは複数のリンクを持つファイルの削除がうまくいきません。そのため、**rm** コマンドを使用して下さい。

```
OS9:rm temp

OS9:ls -e
  0.0   84/02/18 16:23  ----r-wr   1    371       10f Motd
  0.0   84/02/18 16:23  ----r--r   1    374       c7d Errmsg
  0.0   84/02/18 16:23  ----r--r   1    382        11 Dictionary
  0.0   84/11/29 17:17  ---wr-wr   2    384      6c86 Tankanji
  0.0   84/07/02 20:30  -------r   1    3f2        8b password
  0.0   84/11/29 17:17  ---wr-wr   2    384      6c86 temp2
```

```
/*    ln  --  link file to another one    */

/*    usage: ln fromfile tofile.    */


#include <stdio.h>
#include <modes.h>
#include <os9.h>

#define YES 1
#define NO  0
#define ERR (-1)


struct {
    char dtp;
    char dum1[19];
    char fd[3];
    char dum2[9];
} pd;

struct {
    char att;
    int  own;
    char date[5];
    char lnk;
    long siz;
    char cret[3];
} fdes;

char dir[32];
char *newfile;


main( argc, argv )
    int  argc;
    char **argv;
{
    int  pn;
    char *sepname(), *shapename();

    if ( argc != 3 ) {
        fprintf( stderr, "usage: ln fromfile tofile.¥n" );
        return;
    }
    if ( (pn=open(argv[2],0)) != ERR || (pn=open(argv[2],0x80)) != ERR ) {
        close( pn );
        exit( 218 );    /* file already exist */
    }
    if ( chdir( sepname( argv[2] ) ) == ERR )
        exit( errno );
    if ( linkfdes( argv[1] ) == ERR )
        exit( errno );
    if ( ( pn = open( ".", S_IFDIR+S_IREAD+S_IWRITE ) ) == ERR ) {
        fprintf( stderr, "can't open ¥"%-s¥".¥n", shapename( argv[2] ) );
        return;
    }
    dirmake( dir, newfile );
    if ( direntry( pn ) == ERR || write( pn, dir, sizeof(dir) ) == ERR ) {
        close( pn );
        exit( errno );
    }
    close( pn );
}



char *shapename( p )
    char p[];
{
    int  i;

    for ( i = 0; i < 29; i++ )
        if ( p[i] & 0x80 ) {
            p[i] &= 0x7f;
            p[i+1] = '¥0';
            return( p );
        }
    p[29] = '¥0';
    return( p );
}


int direntry( pn )
    int  pn;
{
    static char dir[32];
    long lseek();
    int  n;

    lseek( pn, 64L, 0 );
    while ( n = read( pn, dir, sizeof(dir) ) ) {
        if ( n == ERR )
            return( ERR );
        else if ( *dir == '¥0' ) {
            lseek( pn, -32L, 1 );
            break;
        }
    }
    return( NO );
}


char *sepname( p )
    char *p;
{
    char *newdir, *rindex();

    if ( newfile = rindex( p, '/' ) ) {
        *newfile++ = '¥0';
        if ( *p == '¥0' )
            newdir = ".";
        else
            newdir = p;
    } else {
        newdir = ".";
        newfile = p;
    }
    return( newdir );
}


dirmake( dir, p )
    char dir[], *p;
{
    int  i;

    for ( i = 0; i < 32; i++ )
        dir[i] = '¥0';
    for ( i = 0; i < 29 && ( dir[i] = *p ); p++, i++ )
        ;
    dir[i-1] |= 0x80;
    dir[29] = pd.fd[0];
    dir[30] = pd.fd[1];
    dir[31] = pd.fd[2];
}


int linkfdes( p )
    char *p;
{
    union {
        long lpos;
        char cpos[4];
    } fdpos;
    long lseek();
    int  pn;
    if ( ( pn = open( p, S_IREAD ) ) == ERR )
        return( ERR );
    if ( getstat( SS_OPT, pn, &pd ) == ERR ) {
        close( pn );
        exit( errno );
    }
    close( pn );
    fdpos.cpos[0] = pd.fd[0];
    fdpos.cpos[1] = pd.fd[1];
    fdpos.cpos[2] = pd.fd[2];
    fdpos.cpos[3] = 0;
    if ( ( pn = open( "@", S_IREAD+S_IWRITE ) ) == ERR )
        return( ERR );
    if ( 1seek(pn,fdpos.lpos,0)==ERR || read(pn,&fdes,sizeof(fdes))==ERR ) {
        close( pn );
        return( ERR );
    }
    fdes.lnk++;
    if ( lseek(pn,fdpos.lpos,0)==ERR || write(pn,&fdes,sizeof(fdes))==ERR ) {
        close( pn );
        return( ERR );
    }
    close( pn );
    return( NO );
}
```

## rm

```
/*    rm  --  remove file(s)    */

/*    usage: rm (file).    */


#include <stdio.h>
#include <modes.h>
#include <os9.h>

#define YES 1
#define NO  0
#define ERR (-1)


struct {
    char dtp;
    char dum1[19];
    char fd[3];
    char dfd[3];
    long dcp;     /* directory entry offset */
    char dvt[2];
} pd;

struct {
    char att;
    int  own;
    char date[5];
    char lnk;
    long siz;
    char cret[3];
} fdes;

char root[32];


main( argc, argv )
    int  argc;
    char **argv;
```

```
{
    long lseek();
    int  rpn, pn, clnk;
    char *p, *getroot(), *getdir();

    while ( --argc ) {
        if ( ( rpn = open( getroot( root, p ), S_IREAD+S_IWRITE ) ) == ERR ) {
            fprintf( stderr, "can't open ¥"%-s¥".¥n", root );
            return;
        }
        if ( ( pn = open( p=*++argv, S_IWRITE ) ) == ERR ) {
            fprintf( stderr, "can't open ¥"%-s¥".¥n", p );
            return;
        }
        if ( ( clnk = unlnkfdes( rpn, pn ) ) == ERR ) {
            close( pn );
            fprintf( stderr, "can't unlink ¥"%-s¥".¥n", p );
            exit( errno );
        }
        close( pn );
        close( rpn );
        if ( fdes.lnk == 0 ) {
            if ( unlink( p ) == ERR )
                exit( errno );
        } else {
            if ( ( pn = open( getdir(root,p), S_IFDIR+S_IWRITE ) ) == ERR ) {
                fprintf( stderr, "can't open ¥"%-s¥".¥n", root );
                return;
            }
            if ( lseek(pn,pd.dcp,0) == ERR || write(pn,"¥0",1) == ERR ) {
                fprintf( stderr, "can't delete direcory entry.¥n" );
                close( pn );
                return;
            }
            close( pn );
        }
    }
}



char *getroot( root, p )
    char *root, *p;
{
    int  c, i;

    if ( *p == '/' ) {
        *root = '/';
        for ( i = 1; i < 30 && ( c = *p++ ) && c != '/'; i++ )
            root[i] = c;
        root[i] = '@';
        root[i+1] = '¥0';
    } else
        strcpy( root, "@" );
    return( root );
}



char *getdir( root, p )
    char *root, *p;
{
    char *q;

    if ( q = rindex( p, '/' ) ) {
        if ( p != q )
            *q = '¥0';
        strcpy( root, p );
    } else
        strcpy( root, "." );
    return( root );
}



int unlnkfdes( rpn, pn )
```

```
    int  rpn, pn;
{
    union {
        long lpos;
        char cpos[4];
    } fdpos;

    if ( getstat( SS_OPT, pn, &pd ) == ERR )
        return( ERR );
    fdpos.cpos[0] = pd.fd[0];
    fdpos.cpos[1] = pd.fd[1];
    fdpos.cpos[2] = pd.fd[2];
    fdpos.cpos[3] = 0;
    if ( lseek(rpn,fdpos.lpos,0)==ERR || read(rpn,&fdes,sizeof(fdes))==ERR )
        return( ERR );
    if ( --fdes.lnk ) {
        if (lseek(rpn,fdpos.lpos,0)==ERR||write(rpn,&fdes,sizeof(fdes))==ERR)
            return( ERR );
    }
    return( fdes.lnk & 0xff );
}
```

## find

本文中の find よりいくらか機能を拡張してあります。ファイルの探索の条件として、ユーザ番号および日付を指定することができます。バックアップファイルを作る時に便利です。

```
/*    find  -- find file    */

/*    usage: find <opt> file <opt>.
        opt: -r=DIR   start directory.
             -a       all.
             -u=1     user id.
             -d=yy/mm/dd/hh/mm after the date&time.    */

#include <stdio.h>

#define YES 1
#define NO  0
#define ERR (-1)

int  all = NO;
int  done = NO;
int  group = ERR;
int  user = ERR;
char date[5];
char *root;
char pathlist[BUFSIZ];
FILE *atfp;


main( argc, argv )
    int  argc;
    char **argv;
{
    FILE *fopen();
    char *p;

    if ( option( argc, argv ) == ERR ) {
        usage();
        return;
    }
    while ( --argc ) {
        if ( *(p=*++argv) == '-' )
```

```
                continue;
            if ( root )
                chdir( root );
            if ( ( atfp = fopen( "@", "r" ) ) == NULL ) {
                fprintf( stderr, "can't open ¥'@¥'.¥n" );
                return;
            }
            find( p );
            fclose( atfp );
        }
}


usage()
{
        fprintf( stderr, "usage: find <opt> file <opt>.¥n" );
        fprintf( stderr, "  opt: -r=DIR    start directory.¥n" );
        fprintf( stderr, "       -a        all.¥n" );
        fprintf( stderr, "       -u=1      user id.¥n" );
        fprintf( stderr, "       -d=yy/mm/dd/hh/mm after the date&time.¥n" );
}


int option( argc, argv )
        int  argc;
        char **argv;
{
        char *p, *q;
        int  c, i;

        while ( --argc ) {
            if ( *(p=*++argv) != '-' )
                continue;
            switch ( c = *++p ) {
            case 'r':
            case 'R':
                root = ( *++p == '=' )? ++p: p;
                break;
            case 'a':
            case 'A':
                all = YES;
                break;
            case 'u':
            case 'U':
                p = ( *++p == '=' )? ++p: p;
                if ( q = index( p, '.' ) ) {
                    if ( *p != '.' )
                        group = atoi( p );
                    if ( *++q != '¥0' )
                        user = atoi( q );
                } else {
                    group = ( (c=atoi(p)) >> 8 ) & 0xff;
                    user = c & 0xff;
                }
                break;
            case 'd':
            case 'D':
                if ( strlen( p=(*++p=='=')?++p:p ) == 14 )
                    for ( i = 0; i < 5; i++, p += 3 )
                        date[i] = (*p-'0')*10 + (p[1]-'0');
                else
                    return( ERR );
                break;
            default:
                return( ERR );
            }
        }
        return( NO );
}


find( p )
        char *p;
```

```
{
    FILE *fp, *fopen();
    char *shapename(), *ptappend(), *ptdelete();
    static char dir[32];
    static union {
        long inode;
        char point[4];
    } ip;

     if ( ( fp = fopen( ".", "d" ) ) == NULL )
         return;
     fread( dir, sizeof(dir), 1, fp );
     fread( dir, sizeof(dir), 1, fp );
     while ( fread( dir, sizeof(dir), 1, fp ) ) {
         if ( *dir == '¥0' )
             continue;
         ip.point[0] = dir[29];
         ip.point[1] = dir[30];
         ip.point[2] = dir[31];
         ip.point[3] = 0;
         if ( match( shapename( dir ), p ) && elsematch( ip.inode ) ) {
             putpath( root, pathlist, dir );
             if ( !all ) {
                 done = YES;
                 break;
             }
         }
         if ( chdir( dir ) != ERR ) {
             ptappend( pathlist, dir );
             find( p );
             ptdelete( pathlist );
             chdir( ".." );
         }
         if ( done )
             break;
     }
     fclose( fp );
}


char *shapename( p )
     char *p;
{
     int  i;

     for ( i = 0; i < 29; i++ )
         if ( p[i] & 0x80 ) {
             p[i] &= 0x7f;
             p[i+1] = '¥0';
             return( p );
         }
     p[29] = '¥0';
     return( p );
}


putpath( root, pathlist, dir )
     char *root, *pathlist, *dir;
{
     if ( *root ) {
         fputs( root, stdout );
         putchar( '/' );
     }
     if ( *pathlist ) {
         fputs( pathlist, stdout );
         putchar( '/' );
     }
     puts( dir );
}


char *ptappend( pathlist, p )
     char *pathlist, *p;
{
    if ( *pathlist ) {
        strcat( pathlist, "/" );
        strcat( pathlist, p );
    } else
        strcpy( pathlist, p );
    return( pathlist );
}


char *ptdelete( pathlist )
    char *pathlist;
{
    char *p;

    *( (p=rindex(pathlist,'/'))? p: pathlist ) = '¥0';
    return( pathlist );
}


int elsematch( inode )
    long inode;
{
    static struct {
        char att, own[2], dat[5], lnk;
    } fd;
    int  i, d1, d2;

    fseek( atfp, (long)inode, 0 );
    if ( fread( &fd, sizeof(fd), 1, atfp ) == NULL )
        return( NO );
    if ( group != ERR && group != (fd.own[0]&0xff) )
        return( NO );
    if ( user != ERR && user != (fd.own[1]&0xff) )
        return( NO );
    if ( date[0] )
        for ( i = 0; i < 5; i++ )
            if ( (d1=date[i]&0xff) < (d2=fd.dat[i]&0xff) )
                return( YES );
            else if ( d1 > d2 )
                return( NO );
    return( YES );
}
```

著　者　金井　　隆　　TAKASHI　KANAI
　　　　箕原　辰夫　　TATSUO　MINOHARA

発行者　牧谷秀昭
発行所　秀和システムトレーディング株式会社
　　　　郵便番号１０７　東京都港区南青山２－１９－５　関原ビル

SHUWA SYSTEM TRADING CO., LTD.
SEKIHARA BLDG,
2-19-5 MINAMIAOYAMA, MINATO-KU, TOKYO, 107 JAPAN

印刷所　東京スガキ印刷株式会社

発行日　１９８４年１２月１８日

秀和システム

秀和システムトレーディング株式会社 定価2,300円
ISBN4-87966-032-9 C0000 ¥2300E